JSC Guest Student Programme Proceedings 2011

Edited by Mathias Winkel

FZJ-JSC-IB-2011-06

JÜLICH
FORSCHUNGSZENTRUM

Proceedings 2011

JSC Guest Student Programme on Scientific Computing

Editor
Mathias Winkel

December 2011

FZJ-JSC-IB-2011-06

Jülich Supercomputing Centre
Forschungszentrum Jülich GmbH
D-52425 Jülich

Tel: +49 2461 61-6402
Fax: +49 2461 61-6656
Email: jsc@fz-juelich.de
WWW: http://www.fz-juelich.de/ias/jsc/

FZJ-JSC-IB-2011-06

JSC Guest Student Programme on Scientific Computing
Proceedings 2011

The complete volume as well as reports from earlier Guest Student Programmes are freely available on http://www.fz-juelich.de/ias/jsc/gsp

Editorial

As one of the leading HPC centres in Europe, the Jülich Supercomputing Centre hosts regular support activities and educational programmes in the field of Scientific Computing. One of the main priorities of these events is to introduce young academics to HPC and its applications in scientific research. Therefore, since the year 2000, each summer between eight and twelve international students have had the opportunity to tackle exciting and challenging scientific projects in the field of HPC under the supervision of scientists from JSC, the NIC research group, and other institutes at Forschungszentrum Jülich during the traditional ten week Guest Student Programme on Scientific Computing.

In 2011, the 12th JSC Guest Student Programme took place from 01 August – 07 October and was organized with support from the Centre Européen de Calcul Atomique et Moléculaire (CECAM) and in cooperation with the German Research School for Simulation Sciences (GRS).

Again, the number of excellent international applicants for the programme vastly exceeded the programmes capacity. Finally, twelve students from Germany, Italy, Croatia, Israel, and India joined scientists from JSC, NIC, and GRS for ten weeks. While their original scientific areas ranged from Physics, Mathematics, Chemistry, and Computer Science, their projects covered simulation, visualisation, and algorithm development as well as hardware porting studies.

The guest students and their advisers were:

Sandra Ahnen	(Karlsruhe)	Dirk Brömmel (JSC)
Alexander Alperovich	(Tel Aviv)	Bernhard Steffen (JSC)
Sebastian Banert	(Chemnitz)	Godehard Sutmann (JSC), René Halver (JSC)
Kaustubh Bhat	(Aachen)	Erik Koch (GRS)
Janine George	(Aachen)	Thomas Müller (JSC)
Christian Heinrich	(Cologne)	Bernd Mohr (JSC), Brian Wylie (JSC)
Momchil Ivanov	(Leipzig)	Thomas Neuhaus (JSC)
Andreas Lücke	(Paderborn)	Martin Müser (NIC)
Hans Peschke	(Dresden)	Lukas Arnold (JSC)
Francesco Piccolo	(Napoli)	Herwig Zilken (JSC)
Fabio Pozzati	(Bologna)	Dirk Pleiter (JSC), Willi Homberg (JSC)
Petar Sirkovic	(Zagreb)	Oliver Bücker (JSC), Timo Dickscheid (INM-1)

Besides the intensive use of JSC's reliable workhorses JUGENE and JUROPA, the newly installed GPU cluster JUDGE also played an important role this time: several projects were dedicated to GPU-based algorithms. Therefore, the traditional courses on distributed and shared memory parallel programming and performance optimization were complemented by a workshop on exploiting the capabilities of graphics processing units. During a concluding colloquium, the guest students presented and discussed their results encountered problems and solutions with other students and scientists. This JSC publication contains their individual scientific reports and underlines their ability of self-contained, focused and cooperative work as young scientists on up-to-date topics.

Of course, success of the programme is not only due to single persons but the result of the hard work of many contributors. We would like to thank the guest students for their work and dedication, contribut-

ing to challenging and exciting scientific topics as well as the advisers for their cooperation and patient help, not only regarding the student's work. Thanks go to the lecturers Florian Janetzko, Alexander Schnurpfeil, Jan Meinke, Willi Homberg, Bernd Mohr, and Michael Hennecke who organised and held the training courses.

Additionally, we would especially like to thank Natalie Schröder who has been doing a great job as supporting organizer and Ria Schmitz for her tireless work on and against bureaucracy. Our special thanks go to the Verein der Freunde und Förderer des Forschungszentrums Jülich and to IBM Germany for their support.

Further information, reviews, former results, and the recent announcement of the next programme in 2012 are available online at http://www.fz-juelich.de/ias/jsc/gsp.

Jülich, December 2011 *Mathias Winkel*

Contents

Dynamic load balancing in JuSPIC using MPI/SMPSs

Sandra Ahnen

Karlsruhe Institute of Technology
Faculty for Chemistry and Biosciences
Kaiserstraße 12
76131 Karlsruhe

E-mail: sandra.ahnen@student.kit.edu

Abstract:
Up to now, load balancing in the plasma simulation code JuSPIC has been achieved on two levels: resulting from a domain decomposition of the complete simulation volume, which is necessary for the MPI parallelisation and via distribution of SMPSs tasks onto threads using the new hybrid MPI/SMPSs programming model. During this project, so called dynamic load balancing as a third level has been added. In a first small example, this lead to a considerable speed-up of the program.

---

1 Introduction

The EU project TEXT (Towards EXascale applicaTions) tests and develops with several applications the new hybrid MPI/SMPSs programming model and the dynamic load balancing that is possible with it. The aim is to exploit the full potential of future computers using hundreds of thousands of multicore chips. As part of the project, Jülich Supercomputing Centre ports the Plasma Simulation Code (PSC), already using MPI parallelism, to SMPSs. The goal is to identify necessary improvements of the programming model and to investigate the potential of SMPSs.

2 SMPSs

SMP Superscalar (SMPSs) is a programming model for shared memory parallelism currently developed at the Barcelona Supercomputing Center. The programming environment consists of a source to source compiler and a runtime library and is available for Fortran and C.

The general idea is to execute tasks in threads running on the different cores of a processor, similar to OpenMP. In contrast to OpenMP, using SMPSs the user does not need to assign explicitly which instructions shall be executed in parallel. Instead this is taken care of by the runtime environment based on the data dependencies of the tasks to be run. The user only needs to provide enough tasks of sufficient coarseness that may run in parallel (their execution should take approximately 50-100 μs).

In Fortran, tasks are subroutines with a `!$CSS TASK` annotation in front of the subroutine definition. In addition an interface block is required in the (sub-)program calling the task, specifying the direction of the variables via `intent(in/inout/out)` statements. Starting and stopping the runtime and thus parallel tasks are achieved by adding `!$CSS START` and `!$CSS FINISH` annotations, which must appear only once in the program. For example:

```
program main
  ...
  interface
  !$CSS TASK
  subroutine mytask(a,b,c)
    implicit none
    integer, intent(in) ::  a(1),b(1)
    integer, intent(inout) ::  c(1)
  end subroutine
  end interface
  ...
  !$CSS START
  ...
  call mytask(a,b,c)
  ...
  !$CSS FINISH
  ...
end program main

!$CSS TASK
subroutine mytask(a,b,c)
   ...
end subroutine mytask
```

When compiling the source code, first a source to source compiler is invoked which turns the `!$CSS` annotations into additional source code. At runtime, a task dependency graph is created according to the `intent(...)` statements in the interfaces and the memory regions used.

SMPSs works with different kinds of threads: The main thread populates the task dependency graph and distributes the tasks to the worker threads, which execute them. When there is no work left to distribute, the main thread behaves like a worker thread itself. The user does not need to assign these two kinds of threads. However it is possible to explicitly address a communication thread by `!$CSS TASK TARGET(COMM_THREAD)`. In this way, it is possible to execute communication tasks overlapping with the other tasks.
On Juropa, this can be done using simultaneous multithreading (SMT): Every core of a Juropa compute node possesses two hardware threads, i.e. it can execute two instructions. As these instructions have to physically share the core, SMT is only useful if these instructions are different, e.g. one does calculations and the other one accesses the memory. The communication (reading from or writing to memory) can thus be executed on the second hardware thread of a core.

3 JuSPIC

JuSPIC (Jülich Superscaling Particle-In-Cell code) is based on the Plasma Simulation Code (PSC) originally by H. Ruhl and now developed further at Jülich Supercomputing Centre.
It is used to simulate laser plasma interactions via classical particle simulations. For every simulated time step, the electric and magnetic fields and the motion of the particles of the plasma need to be computed. For this purpose a domain decomposition is applied to distribute the complete simulated volume onto the MPI processes. The electric and magnetic fields are located on gridpoints, the particles of the plasma have continuous coordinates inside the local domains. Communication between the local domains is necessary to be able to continuously calculate the fields and to account for particles leaving one domain and moving on to the neighbouring one.
Within the TEXT project, most of the simulation steps for the local domains have been put into SMPSs tasks.

4 Dynamic load balancing

Up to now, load balancing in JuSPIC had been achieved on two levels: resulting from a domain decomposition of the complete simulation volume, which is necessary for the MPI parallelisation and via distribution of SMPSs tasks onto threads. During this project, dynamic load balancing as a third level has been added.

On the first level of load balancing, the total work load is distributed onto MPI processes. For this domain decomposition a mesh with varying width of the sections is used (Fig. 1). Local domains containing plasma are chosen to be smaller than those without. This accounts for the additional amount of work to simulate the particles of the plasma. In the used mesh, the width is set for each line and column, variation of the width within one line/column is not possible. Therefore, the mesh is not flexible enough to distribute the work equally in all cases: In Fig. 1 domain 3 contains no plasma and only needs to compute the electric and magnetic fields in a large volume. The volume of domain 2 is smaller but there is additionally plasma to be simulated. While the total work load of these two domains might be equal, domain 1 in contrast is small and contains no plasma. The corresponding MPI process will therefore have less work. Since in each simulation step communication between the domains is necessary, it will have to wait for the others to finish and be idle during this time.

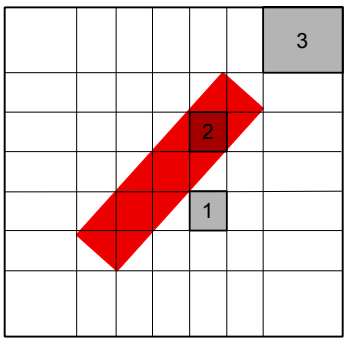


Figure 1: Example mesh for domain decomposition for a plasma lying diagonally in a 2D simulation box. The three light coloured boxes indicate three domains with different work.

Inside of one local domain, the second level of load balancing takes place: the work in form of coarse grained tasks is distributed automatically among the SMPSs threads. If one thread finished its tasks, it may steal work from another thread, so that no thread (i.e. no core of the compute node) is idle.

Hybrid MPI/SMPSs makes it possible to use dynamic load balancing as a third level. The idea is that MPI processes in a blocking communication or waiting at a barrier give the cores occupied by their SMPSs threads to another MPI process on the same compute node. The other MPI process can then extend its SMPSs tasks to more threads and might therefore finish earlier. This shuffling of cores is realised by an extra runtime library. The library intercepts all calls to MPI functions and decides if the call blocks the thread and thus if dynamic load balancing can take place.

Back in Fig. 1, the MPI process corresponding to domain 1 could then give its cores to the MPI process of domain 2 or 3, as long as they are located on the same compute node. The idle time might thus be reduced resulting in an overall speed-up of the program.

In the Paraver trace of a dummy example (Fig. 2), the dynamic load balancing (DLB) can be observed: The first two windows show the tasks (dark and white boxes) executed by the different threads on the same timescale. In this case, the upper half of the threads belong to the first MPI process, the lower half to the second. The third window shows which MPI process is working or in a blocking MPI communication. Without DLB, every MPI process works on two threads (first window). Using DLB (second window), the overall execution time is shorter and the MPI processes sometimes work on four threads instead of two. Comparing with the third window, one can see, that this is exactly the case, when one MPI process is in a blocking communication and the other one is working. The working process gets the cores of the blocked one and can therefore distribute its SMPSs tasks onto four threads instead of two.

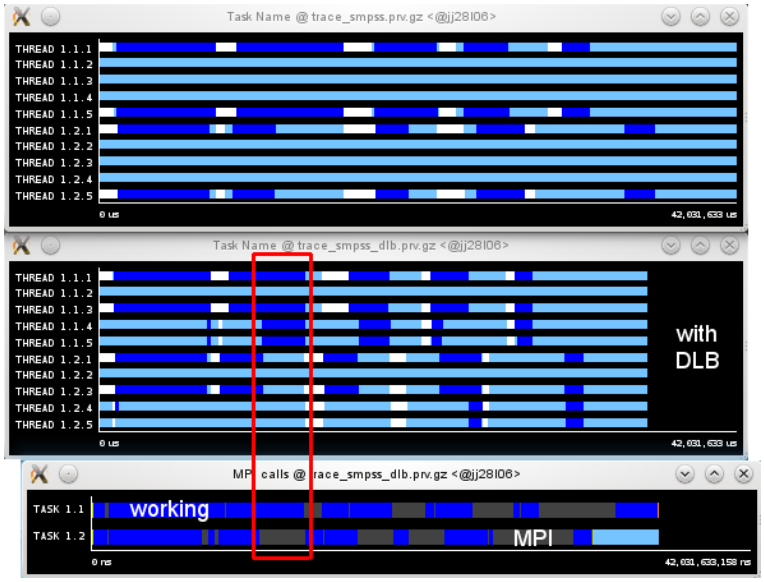


Figure 2: Paraver screenshot: The same application run without (upper window) and with dynamic load balancing (middle window) with the same timescale. The third window shows when an MPI process is working or in a blocking MPI communication. The box indicates an area, where DLB is taking place.

5 Improvements on JuSPIC

The communication inside JuSPIC has been moved into SMPSs tasks. This increases the SMPSs parallelism of the code, because more tasks are available to be distributed onto the threads. With correct data dependencies, the communication will be executed overlapping with the other tasks, using the communication thread mentioned before.

The so far non-blocking MPI communication has been changed to blocking communication. This might not be an advantage by itself, but the blocking MPI_SENDRECV call indicates the runtime to use dynamic load balancing.

In a small example, simulating only ten time steps and running on very few nodes, this lead to a considerable speed-up (see Table 1). The speed-up depends of course on the specific example, i.e. on the work imbalance to start with. In the Paraver trace (Fig. 3), the dynamic load balancing (DLB) can be observed: The first window shows which MPI processes are working and which are in a blocking MPI communication. The second window shows the number of threads used by each process. In this case, two SMPSs threads per MPI process, and two MPI processes per node have been assigned. Only the two processes on the same node can interchange the cores occupied by their SMPSs threads. Accordingly, the use of 4 threads by one MPI process indicates dynamic load balancing. The appearance of dynamic load balancing always corresponds to one process working and one being idle (compare first and second window). The third window displays the tasks executed on the SMPSs threads. The performance could still be improved allowing more or different MPI tasks to interchange their cores.

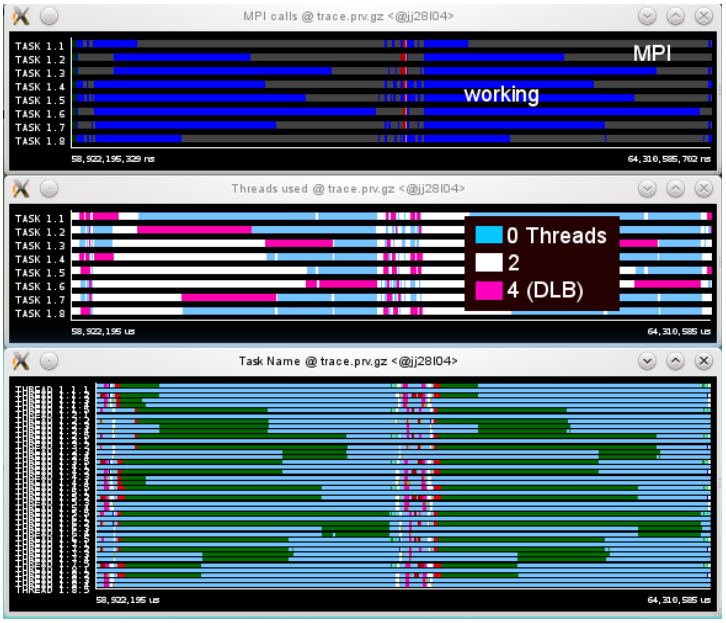


Figure 3: Paraver screenshot of JuSPIC: The first window shows which MPI processes are working and which are in a blocking MPI communication. The second window shows the number of threads used by each process. In this case, two SMPSs threads per MPI process, and two MPI processes per node have been assigned. Accordingly, the use of 4 threads indicates dynamic load balancing. The third window displays the tasks executed on the SMPSs threads.

		total Walltime in s	speed-up
4 nodes with each	w/o DLB	149.07928	
2 MPI x 2 SMPSs	w DLB	139.05904	7 %
2 nodes with each	w/o DLB	282.75693	
4 MPI x 1 SMPSs	w DLB	237.75237	16 %

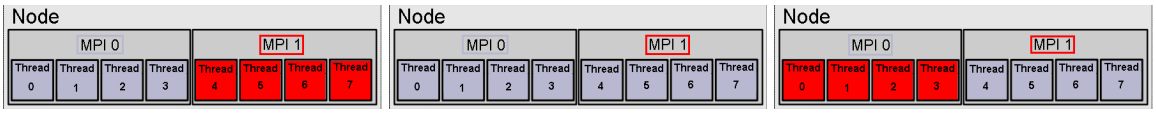
Table 1: Performance measurements of JuSPIC on Juropa with and without dynamic load balancing (DLB).

6 Outlook

Up to now, JuSPIC does not clearly define all data dependencies and hence the order of the tasks to be run. To ensure the correct execution of the program, barriers are inserted. Unfortunately, these inhibit the runtime environment to generate the task dependencies and therefore to schedule the tasks to the best of its abilities. This disadvantage could be removed by inserting sentinels. Sentinels are dummy variables committed to the tasks in such a way, that via their `intent(...)` statements, the order is non-ambiguously defined.

In order to profit more from the dynamic load balancing, it could be tried to distribute the MPI processes in such a way, that processes with different work load end up on the same compute node. The work could then be balanced dynamically more evenly on the cores and the idle time could be reduced.

Furthermore, the dynamic load balancing revealed some problems: Consider two MPI processes with each 4 SMPSs threads executed on one compute node (Fig. 4a). If MPI process 1 gives its cores to MPI process 0 (Fig. 4b), the threads on these cores are already a bit slower than those on the socket, where MPI 0 is located itself. The reason for this is that the two sockets have separate memory, so the threads on the lent cores need to access the memory on the other socket, since they belong to MPI process 0. If then MPI process 1 gets four cores back, these are not necessarily the same cores it gave away. It simply gets those back, that finished their work first. In the worst case, the threads of MPI process 1 might end up completely on the socket of MPI 0 (Fig. 4c). This way, the performance after interchanging the cores would be much poorer than before any dynamic load balancing happened.
The work-around up to now is to use only one socket of each node in order to access the same memory no matter how the cores are interchanged. This way, half of the node is unused, which is of course undesirable. A better way might be to restrict the dynamic load balancing to the MPI processes located on the same socket.



(a) (b) (c)

Figure 4: Problems with DLB: Schematic picture of a Juropa compute node, occupied by 1 MPI process per socket each using 4 SMPSs threads, before (a), during (b) and after DLB (c).

7 Acknowledgements

I would like to thank my adviser Dirk Brömmel for introducing me to the interesting, even if sometimes a bit curious world of SMPSs, and for always coming round to my office with his cup of tea. I had fun and learned a lot.
Further thanks go to Mathias Winkel, Natalie Schröder and everyone else involved in organizing the Guest Student Programme.

References

1. SMP Superscalar (SMPSs) User's Manual, Version 2.4, Barcelona Supercomputing Center, July 2011.
2. Barcelona Supercomputing Center, SMPSs, http://www.bsc.es/plantillaG.php?cat_id=385
3. TEXT Project, http://www.project-text.eu/
4. H. Ruhl, Classical Particle Simulations with the PSC code.

Support for performance measurements of MPI File I/O for the Scalasca toolset

Christian Heinrich

Universität zu Köln
Mathematisches Institut
Weyertal 86-90
50931 Köln

E-mail: heinrich@informatik.uni-koeln.de

Abstract:
Scalasca is a portable and scalable performance measurement tool that provides sophisticated support for MPI. Unfortunately, support for the performance measurements of MPI File I/O was limited, for example it did not provide any facility to get information about the amount of bytes transferred (read/written) in a specific MPI File I/O call.
In this report, the steps that have been taken to implement this will be described. Additionally, an introduction into the Scalasca wrapper generator will be given.

---

1 Introduction

Parallel programming becomes more and more important as computational sciences require a growing amount of resources since their computations quickly become more and more complex. However, this performance gain cannot be achieved by simply increasing the CPU core speed, as the latter is limited due to exponential growth in needed energy and produced heat. Instead, the amount of cores is increased and new programming paradigms have been developed so that programs run on multiple cores and processors at the same time.

Unfortunately, parallel programming is hard - it is very easy to introduce deadlocks and performance bottlenecks which might be absolutely non-obvious and cause dramatic performance loss.

Some examples for this could be:

1. Heavy communication between nodes that is not necessary or could at least be reduced. Communication over physical networks is especially expensive due to the comparatively slow speed.
2. Work loads may easily be incorrectly partitioned. This means that several processes are given considerably more work while others are idle.

3. Computational dependencies might be set up inefficently. This means that several processes have to wait for another process to finish its computation as they need the results to continue their own work.

The goal of the field of ”performance measurement and analysis“ is to find these bottlenecks or at least to provide information so that the user can find them on his own.

There are several tools available that support the user to do so. Examples are Vampir[1], TAU[2] as well as Scalasca[3]. Only Scalasca will be used in this report.

2 Scalasca

Scalasca[1] is an automatic performance evaluation system for MPI, OpenMP, and hybrid applications written in C/C++ or Fortran. Scalasca generates event traces from running applications and automatically searches them off-line for execution patterns indicating inefficient performance behavior. Scalasca is jointly developed by Forschungszentrum Jülich, Germany, and the German Research School for Simulation Sciences in Aachen.

The neccessary instrumentation of user code is supported in different ways depending on the availability of certain compilers and third-party tools: automatically using a compiler-supplied monitoring interface or using TAU, or manually by placing POMP directives after the entry point and before all exit points of arbitrary user-defined regions. The POMP directives are later processed by OPARI, which is also responsible for the automatic instrumentation of OpenMP constructs. MPI point-to-point, collective, and one-sided communication and synchronization functions are instrumented fully automatically by interposing a wrapper library. During execution, the instrumented code generates several trace files, one for each MPI process.

After the measurement has been completed, the traces are subjected to an off-line analysis performed by Scalasca's trace analyzer, which attempts to identify specific performance properties. Internally, it represents performance properties in the form of execution patterns that model inefficient behavior. These patterns are used during the analysis process to recognize, classify, and quantify inefficient behavior in the application. The performance properties addressed by Scalasca include inefficient use of the parallel programming models MPI and OpenMP. The analysis process automatically transforms the traces into a compact call-path profile that includes the execution time penalties caused by the different patterns broken down by call path and process or thread.

2.1 Instrumentation

In this context, the term “instrumentation” can be comprehended as “inserting function calls into an existing programs to collect specific data”. Examples for such data might be timestamps for enter / finish times of a function, parameter values, communication data or maybe just data that helps with debugging.

[1] http://www.vampir.eu
[2] http://tau.uoregon.edu
[3] http://www.scalasca.org

Scalasca's measurement system instruments functions that the user wants to track by either using compiler instrumentation (thus, automatic) or manual instrumention by the user. Automatic instrumentation is most of the time only appropriate for standard measurements like time measurement.

However, for libraries such as MPI where more information ought to be tracked, automatic instrumentation is impossible as it must be determined how the data needed can be collected. To do this, so-called wrappers are used.

As the name suggests, these wrapper functions wrap other functions and thus add functionality. This is shown exemplarily in the following listing:

```
MPI_Send( ... ) {
    begin_measurement();
    returnVal = PMPI_Send(...);
    finish_measurement();
    return returnVal;
}
```

Listing 2.1: Principle structure of an MPI wrapper

This illustrates the way Scalasca wraps MPI functions: MPI already ships with an interface for this purpose called PMPI - every MPI function exists as MPI_Function() as well as PMPI_Function(). This allows to implement wrappers which behave like the original MPI call but introduce extra instrumentation code. This means that the user, who should always call MPI_Function() does not know whether a wrapper is being called or not but since the wrapper does not change behaviour, there is no reason the user should know.

The listing above demonstrates an easy use-case for this: The common function MPI_Send is wrapped and two function calls are issued. One initiates measurement before the original PMPI_Send is executed and the other finishes it afterwards. This is mainly the way that Scalasca tracks time spent in any MPI function. Another feature of the "wrapper" technique is that the instrumentation has access to all parameters passed in and out of the function. To use wrapper libraries, the user has to link first with the wrapper library, then with the original MPI library. Wrapper functions will be the central part of this work.

3 MPI File I/O

Although POSIX (the "Portable Operating System Interface") defines a widely portable file system, efficient parallel I/O is hard to achieve since POSIX lacks support for sophisticated parallel file accesses such as parallel reads.

This motivates the introduction and usage of a standardized and performant, but still optimizable parallel file I/O library. Standardization is important for portability whereas optimizability will allow developers to tweak their programs.

MPI in fact provides such facilities. However, although MPI is quite easy to use, about 90% of programs that use MPI still use standard I/O instead of MPI File I/O.

3.1 Structure of MPI File I/O

positioning	synchronism	coordination	
		noncollective	*collective*
explicit offsets	*blocking*	MPI_FILE_READ_AT MPI_FILE_WRITE_AT	MPI_FILE_READ_AT_ALL MPI_FILE_WRITE_AT_ALL
	nonblocking & split collective	MPI_FILE_IREAD_AT MPI_FILE_IWRITE_AT	MPI_FILE_READ_AT_ALL_BEGIN MPI_FILE_READ_AT_ALL_END MPI_FILE_WRITE_AT_ALL_BEGIN MPI_FILE_WRITE_AT_ALL_END
individual file pointers	*blocking*	MPI_FILE_READ MPI_FILE_WRITE	MPI_FILE_READ_ALL MPI_FILE_WRITE_ALL
	nonblocking & split collective	MPI_FILE_IREAD MPI_FILE_IWRITE	MPI_FILE_READ_ALL_BEGIN MPI_FILE_READ_ALL_END MPI_FILE_WRITE_ALL_BEGIN MPI_FILE_WRITE_ALL_END
shared file pointer	*blocking*	MPI_FILE_READ_SHARED MPI_FILE_WRITE_SHARED	MPI_FILE_READ_ORDERED MPI_FILE_WRITE_ORDERED
	nonblocking & split collective	MPI_FILE_IREAD_SHARED MPI_FILE_IWRITE_SHARED	MPI_FILE_READ_ORDERED_BEGIN MPI_FILE_READ_ORDERED_END MPI_FILE_WRITE_ORDERED_BEGIN MPI_FILE_WRITE_ORDERED_END

Figure 1: Functions for read / write accesses in MPI 2.2. Picture taken from [6] chapter 13.4.1

As can be seen in figure 1, there are many MPI File I/O functions available. These are distinguished as follows:

1. The positioning states where the function will write or read. The position can either be explicit for every rank (“explicit positioning”), it can be performed by individual file pointers (every rank maintains its own file pointer; if this file pointer is changed, other ranks are not affected) or by shared file pointers (every change to this file pointer affects all other ranks participating in this operation as well).
2. Furthermore, synchronism of these functions may differ. There are in general three categories:

   a) Blocking

   b) Non-Blocking

   c) Split Collective

   Blocking means that the function call returns only after the operation completed (or an error occurred).

   Non-Blocking is complementary and means "function can return immediately". Non-Blocking operations are only complete when indicated by a subsequent MPI_Test or MPI_Wait. This can obviously speed up a program since data can be read (or written) in the background while other computations are still being executed.

These two categories are the same as those known from MPI Point-to-Point (P2P) functions.

Additionally, for each blocking collective I/O function, a restricted "non-blocking collective" function is available in form of a so called “split collective” function.

3. Thirdly, there are differences in coordination: There are collective and non-collective functions. These terms are used to state whether this function has to be called by every rank which participated in the “file open”-call associated with the filehandle used (collective) or executes independently (non-collective).

4 My Task

4.1 Motivation

When optimizing a parallel program that uses MPI File I/O, programmers might be interested in the following questions:

- How many files are opened in total?
- How much time is consumed by I/O operations?
- How many bytes does my program read / write in total?
- How many bytes does a specific rank k read?
- Is the amount of bytes read / written equally partitioned or are some ranks doing more work than others?

It is quite obvious that Scalasca is predestined to answer these questions. Unfortunately, Scalasca 1.3.3 only keeps track of the amount of file operations as well as the time they consume - there is no metric that could keep track of the other desired information. Thus, it would be good to have support for this in an upcoming version of Scalasca.

4.2 Task description

Implement everything necessary for Scalasca to track bytes read / written in MPI File I/O operations.

Proceed as follows:

1. Make any necessary changes to the Scalasca core system; this includes adding new functions to the measurement system and a new metric
2. Familiarize yourself with Scalasca’s function wrapper generator.
3. Write thorough documentation that can be used by future developers to understand and use the wrapper generator.
4. Set up additional templates and extend prototype definitions where needed.
5. Test your results by writing several test cases.

5 The wrapper generator

5.1 Motivation

Although the wrapper generator is only of interest for Scalasca developers and not for endusers, it is a vital part of Scalasca since there are currently more than 300 functions for each of C and Fortran in MPI that need to be instrumented to track at least the time spent within them. As manual instrumentation would imply a lot of redundant code, this code gets automatically generated. How this works will be explained in the following paragraphs.

Further reasons why using a wrapper generator might be worthwhile:

1. Fortran code must be instrumented as well, so wrapper for the C and Fortran API of MPI need to be generated; this adds a significant amount of work.
2. Generating avoids error-prone code-copy and reduces the amount of redundant code.
3. It is much faster than writing by hand.
4. Although mainly used for MPI, other libraries could be wrapped as well if the generator is designed properly.

5.2 What does it provide?

The wrapper provides:

1. A storage file (in XML format) for function definitions, called “prototype“ storage, storing function names, parameters and types, return values, etc.
2. High-level templates that are used to decide which functions to wrap (selection) and in which order. (This is implemented by #pragma’s)
3. Low-level templates contain the desired structure of the resulting wrapped function. To keep it general, variables can be used.
4. Abovementioned #pragma statements use these low-level templates and replace the variables with definitions provided by the “prototype“ storage.

5.2.1 Prototype storage

The storage must contain a definition for every function to be wrapped. This includes basic information like function name, return type, parameter lists as well as library specific information like the version that introduced this function. Other tags provide the possibility to declare own variables and initialize them, execute own code, provide rules to compute specific parameters of this function or just to associate the function with a group.

The following excerpt of the storage file shows the definition of “MPI_Isend”.

```
<prototype name="MPI_Isend" rtype="int" group="p2p" guard="p2p" >
```

```
    <param access="i" type="void*" name="buf" />
    <param access="i" type="int" name="count" />
    <param access="i" type="MPI_Datatype" name="datatype" />
    <param access="i" type="int" name="dest" />
    <param access="i" type="int" name="tag" />
    <param access="i" type="MPI_Comm" name="comm" />
    <param access="o" type="MPI_Request*" name="request" />
    <version id="1.0" />
</prototype>
```

Listing 2.2: Definition of MPI_Isend as used in Scalasca

5.2.2 Templates

As aforementioned, the wrapper generator ships with two different kinds of templates: On the one hand, we have the high-level templates which are mainly, but not always, used to decide which functions should be wrapped and in which order. They provide several mechanisms how to select the desired functions or even contain hardcoded functions, if the code is too function specific.

On the other hand, we have the so-called low-level templates: They define the structure of the generated code for the functions that were selected by the high-level templates. As a result, function specific code is generated because low-level templates allow the use of variables (which simply access the data associated with the currently processed function that was stored in the prototype storage). However, it is not the case that the result itself is always code for a function.

A trivial example of a high-level template is shown below, followed by two more listings showing the associated low-level template (listing 2.4) as well as resulting output (listing 2.5). Both templates are probably one of the easiest cases that could be thought of.

```
/** High level template **/
#pragma wrapgen multiple restrict() skel/epk_mpireg_h.w
```

Listing 2.3: This high level template selects all functions contained in the prototype storage and generates code for them with the low-level template skel/epk_mpireg.h.w

```
/** EPIK region ID for ${name} */
#define EPK__${name|uppercase} ${id}
```

Listing 2.4: Content of skel/epk_mpireg_h.w. Here, a constant is defined by using the uppercase helper. This helper transforms the function name stored in $name into uppercase letters.

5.3 Documentation

During the summer program at JSC, documentation for users of this wrapper generator was written. The documentation itself is several pages long and explains the generator in detail. However, this is not possible in this report.

```
/** EPIK region ID for MPI_Abort */
#define EPK__MPI_ABORT                                  0
/** EPIK region ID for MPI_Accumulate */
#define EPK__MPI_ACCUMULATE                             1
/** EPIK region ID for MPI_Add_error_class */
#define EPK__MPI_ADD_ERROR_CLASS                        2
/** EPIK region ID for MPI_Add_error_code */
#define EPK__MPI_ADD_ERROR_CODE                         3
```

Listing 2.5: Generated code from the two preceding listings. Similar lines are generated for all other MPI functions as well.

Exemplarily, it covers the prototype storage in detail as well as both high- and low-level templates. Additionally, background information is given (how does the fortran wrapper work?) and recurring tasks like adding a new variable or the standard workflow are described.

6 Changes

6.1 Changes to Scalasca's core system

At first, the Scalasca core system had to be prepared in order to record and display the bytes transferred. This includes adding new metrics to its EPISODE measurement system. There are a lot of places that had to be adjusted with a few lines of code. These changes were developed in cooperation with the Scalasca team.

6.2 Changes within the wrapper generator

The wrapper generator needed a little bit more work:

1. Create a new template that allows to execute code after the PMPI function was called. This was solved by introducing a new variable "postcall".
2. Extend prototype definitions: Declare and initialize variables that are needed within the "postcall" section. Prototype definitions that had to be extended include all blocking and non-blocking functions (see figure 1). For non-blocking functions, MPI_Wait had to be extended in addition to the respective function prototypes. A typical postcall section for some function prototype might look like listing 2.6.
3. Adjust the "make" process.

7 After the changes

Since the changes were made quickly, tests could be performed.

```
<postcall>
    <![CDATA[
    PMPI_Type_size(datatype, &size);
    PMPI_Get_count(status, datatype, &receivedEntries);
    esd_mpi_file_read( size * receivedEntries );
    ]]>
</postcall>
```

Listing 2.6: Example for a postcall section within the prototype storage

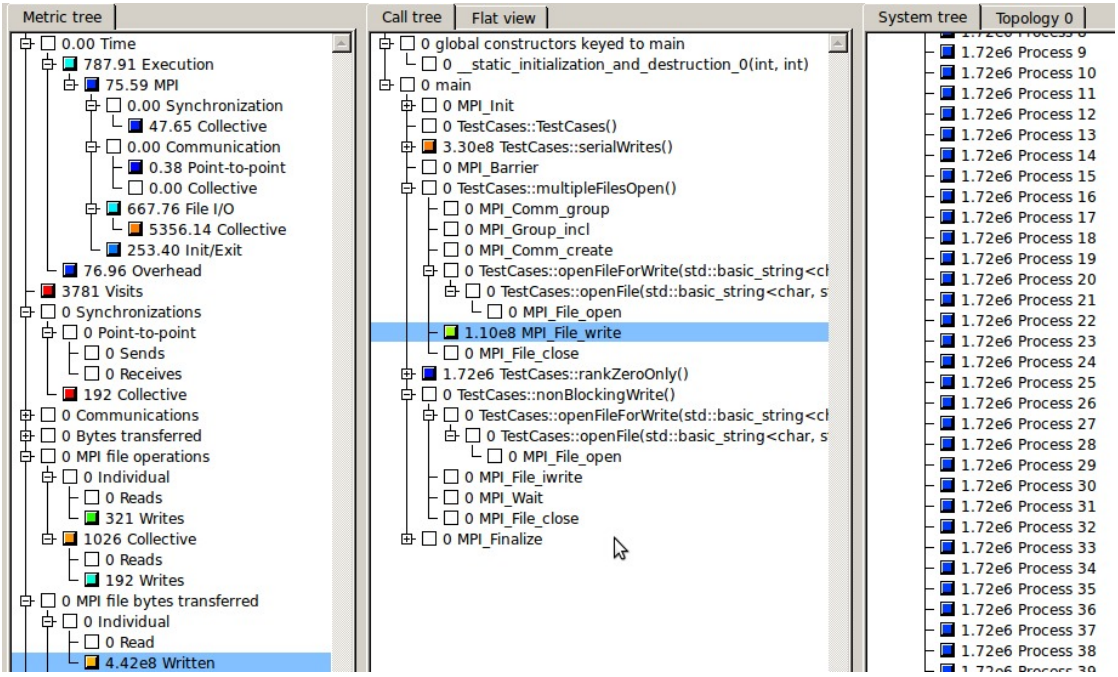


Figure 2: Scalasca analysis report presentation with a new metric for transferred bytes in I/O operations.

In figure 2, a measurement of a testprogram with MPI File I/O is illustrated. In the left column, all metrics are displayed and you can easily spot the new MPI File I/O metrics at the bottom. Since this figure only shows a portion of the Scalasca GUI, the “collective” metrics cannot be seen but are also available.

In the situation shown, the metric “MPI Individual File Bytes Written” is selected and thus, the middle column shows the total amount of bytes written for each function call: For example, only three functions use MPI to write files (“TestCases::serialWrites”, “TestCases::rankZeroOnly” as well as “TestCases::multipleFilesOpen” whereas the last one is divided into further function calls where you can see that only “MPI_File_write” actually writes).

With “MPI_File_write” selected in the middle column, the right column shows the bytes written by each rank by this function call. In this testcase, every rank writes the same amount which is reported correctly.

7.1 Testcases

Multiple testcases have been written and are mostly very basic; in fact, they are only issuing several calls of the form “open - read / write - close” so that it can be checked whether Scalasca tracks everything without error.

Further test cases include:

- Test non-blocking (including split-collective) functions separately from blocking functions
- Check if everything is recorded correctly if only one rank reads / writes
- Open multiple files at the same time - split communicator for this

These tests will be expanded in future as some functionality like specifying the number of bytes to be read / written on startup is still missing.

8 Problems

After testcases for non-blocking functions had been written and tested, it became clear that support for non-blocking functions would require serious changes within Scalasca’s core due to design decisions: Each function was erroneously considered to belong to at most one group - however, MPI_Wait is a counter example as it could be used to complete a P2P or I/O operation. Scalasca, however, associates MPI_Wait only with P2P which implies that P2P operations appear within the GUI even though there were no such operations involved. This can be seen in figure 3.

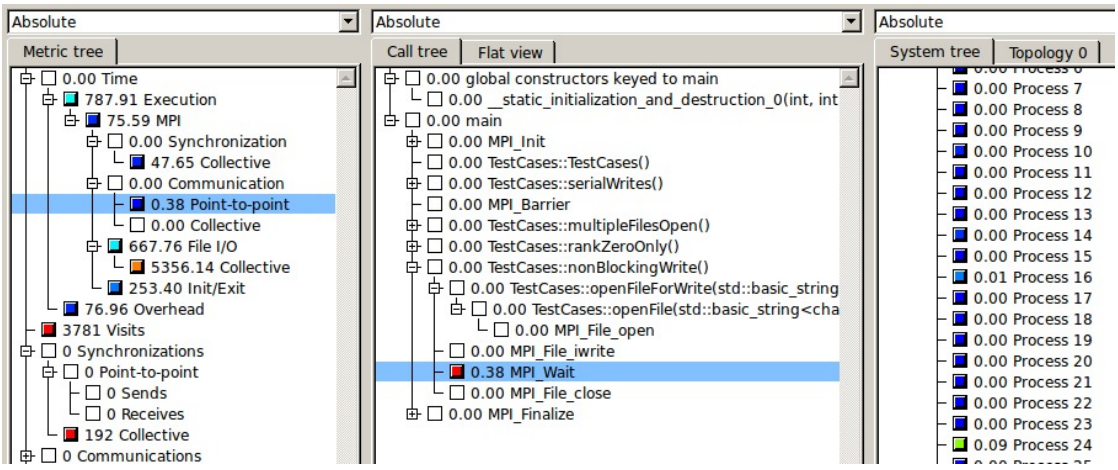


Figure 3: Scalasca shows P2P activity, however, there was none.

9 Conclusion

Scalasca’s measurement and analysis modules were extended to include data about MPI parallel file I/O. Scalasca now allows parallel programmers to analyze and tune the parallel I/O of their code. Still,

there is quite some work to do: Split collective functions can currently not be tracked and Scalasca is unable to distinguish completion of non-blocking and P2P and File I/O operations.

References

1. Markus Geimer and Felix Wolf and Brian J. N. Wylie and Erika Ábrahám and Daniel Becker and Bernd Mohr "The Scalasca performance toolset architecture", in: Concurrency and Computation: Practice and Experience, Vol. 22 No. 6, Pages 702-719. April 2010
2. Message Passing Interface Forum, “MPI Standard Version 2.2”, 2009, High-Performance Computing Center Stuttgart (University of Stuttgart)

Hierarchical Tree Construction in PEPC

Hans Peschke

Dresden University of Technology
Department of Mathematics
Institute of Scientific Computing
01062 Dresden, Germany

E-mail: hans.peschke@mrsep.de

Abstract:
The highly scalable parallel tree-code PEPC is the first Barnes-Hut tree-code implementation which runs efficiently on the entire 288k cores of JUGENE. This is possible as almost all parts of the code scale perfectly up to this amount of cores. The currently problematic code segment handles the global exchange of branch-nodes which is going to dominate the overall run-time for an increasing number of cores. Branch-nodes are essential for the tree-traversal, since they act as entry points to remote trees. The aim of this paper is to describe the scalability issues and to design and implement an algorithm for the hierarchical tree construction in order to optimise the global exchange of data.

1 Introduction

The parallel tree-code PEPC (Pretty Efficient Parallel Coulomb Solver) is an efficient FORTRAN implementation of the Barnes-Hut tree-code [2] based on the Hashed Oct-Tree (HOT) scheme from Warren and Salmon [6] for simulating N-body systems with $N \gg 3$. Although the method of Barnes and Hut reduces the complexity to $\mathcal{O}(N \log N)$, compared to the direct summation with a complexity of $\mathcal{O}(N^2)$, it is also essential to reach very high scalability in order to utilise the full capabilities of current and future supercomputers.

With the emergence of the current petascale supercomputer generation with thousands of cores, PEPC was optimised to achieve very good strong-scaling results [3, section 3] with up to 8192 MPI-tasks. The latest improvements, namely Virtual Local Domains [4] as well as a hybrid parallelized tree-traversal with a fully asynchronous communication scheme [7], pushed up the limit even further so that PEPC now scales on the full IBM Blue Gene/P system JUGENE at Jülich Supercomputing Centre with its 294,912 cores.

These very good results are solely possible due to almost all parts of the code being scaled perfectly. This even applies to the algorithmically most challenging tree-traversal and force-summation, which generally is most time consuming. As we will see in section 2, every parallel Barnes-Hut tree-code has to exchange data between all processes in every time-step. This part of the algorithm is indeed

not expected to scale at all, but its impact has to be reduced to an absolute minimum to be prepared for future supercomputer generations. The objective of this paper is to describe the scalability issues in the current implementation of the parallel Barnes-Hut tree-code PEPC and to design and implement an algorithm for the hierarchical tree construction in order to optimise the global exchange of data.

For the purpose of understanding the current scalability problem in PEPC, which is analysed in section 3, it is necessary to give a short introduction to the basic algorithm of the parallel Barnes-Hut tree-code, as implemented in PEPC. This is dealt with in the following section, mainly focusing on the global exchange of data.

Section 4 presents a new hierarchical tree construction algorithm as a solution for the scalability issue described in section 3. The last two sections show scalability results as well as an outlook to further improvements.

2 The parallel Barnes-Hut tree-code algorithm in PEPC

This section gives an overview of the whole algorithm. Selected parts are described in detail in order to understand the scalability problem as well as the new algorithm presented in section 4. For deeper insight into the algorithm and the implementation, [3] and [5] are good starting points.

Algorithm 1 main steps of the parallel Barnes-Hut algorithm

for all time steps **do**
  step 1: domain decomposition
  step 2: building up of the local tree
  step 3: global exchange of the branch-nodes
  step 4: building up of the global tree
  step 5: tree-traversal and force-summation for each particle
end for

At the beginning of every iteration, the particles are randomly distributed over all processes. In the first step, they therefore have to be redistributed in a manner that does not leave the domains too fragmented for the purpose of gaining good efficiency. In PEPC, this is achieved by parallel linear sorting of the particle keys on a space-filling curve. The *key*s are unique octal numbers representing a node in the hashed oct-tree and can be calculated using the coordinates of the particles.

In the next step, every process builds up the local hashed oct-tree from its local set of particles and their corresponding keys. This is performed from the *root-node* down to the *leafs*. Every *leaf-node* contains exactly one particle and only the nodes between leaf-nodes and the local root-node are stored. After the construction of the local tree has been finished, every process determines a set of nodes which covers the whole local domain, but does not overlap with the domains of other processes. The elements of this set are called *branch-nodes* or *branches*. For efficiency reasons this set needs to be as small as possible. At the end of the local tree-build, every process calculates the multipole-properties of every node in the local tree from the leaf nodes up to the root node.

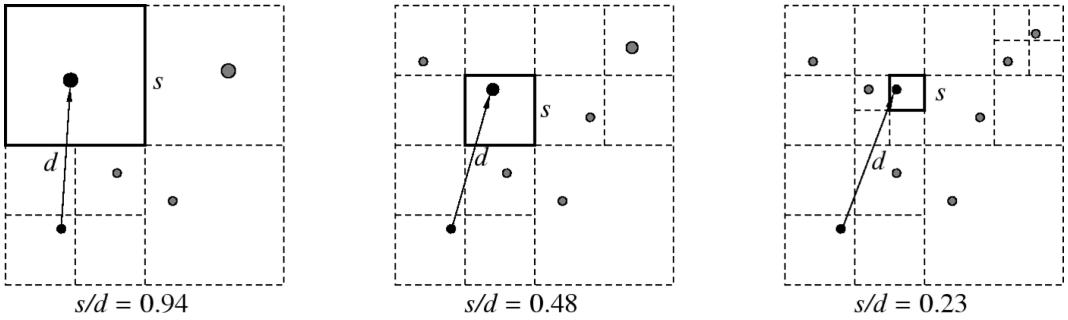


Figure 1: Multipole Acceptance Criterion: Interact with a node if $s/d < \theta$ for a predefined $\theta \in (0, 1]$, otherwise resolve the node and interact with its children. [1]

The branch-nodes are necessary in step 5 of algorithm 1 as they interact with particles or multipoles in remote domains. They store the location (MPI-rank[1] number) of their own child-nodes and act as entry points to remote trees. In order for this to work properly, the branch-nodes of all domains have to be exchanged, meaning that after step 3 in algorithm 1 every local hashed oct-tree of a process contains all branch-nodes of all remote hashed oct-trees in addition to its local branch-nodes. In PEPC, this is performed utilising the MPI-library. While, for each process, `MPI_AllGather` communicates the number of nodes, that have to be exchanged, to every process, `MPI_AllGatherV` exchanges the branch-nodes including their multipole-properties.

Subsequently, every rank calculates the multipole-information of every node above the branches including the root-node in step 4. The last step comprises the tree-traversal and the force-summation. For every local particle, an interaction list of particles and multipoles is determined by the corresponding process according to the *Multipole-Acceptance-Criterion* (MAC), shown in figure 1. If the children of a node n are not available in the local hash-table, which is the case for all non-local branch-nodes, the node n stores their locality, or more precisely their MPI-rank number, so as to be able to request them during the tree-traversal.

According to [3, section 2.6], "the tree-traversal is the most important and algorithmically challenging part of a parallel tree-code". The current implementation uses Pthreads as well as an asynchronous communication scheme to overlap communication and computation. Since the details are not relevant for the rest of this paper, [7] is recommended for further reading.

3 The current scalability issues in PEPC

In order to achieve perfect strong-scaling results with thousands of MPI-ranks, it is imperative that all parts of the algorithm scale well. With an increasing number of processes, non-scaling code segments will dominate the run-time at some point. To be prepared for future supercomputer generations with millions of cores, these non-scaling parts ought to be avoided. The IBM Blue Gene/P system JUGENE at Jülich Supercomputing Centre with its 294,912 cores enables to get an insight into the scalability problems on such massively parallel systems.

[1] The terms MPI-rank, rank, MPI-process and process are used synonymously.

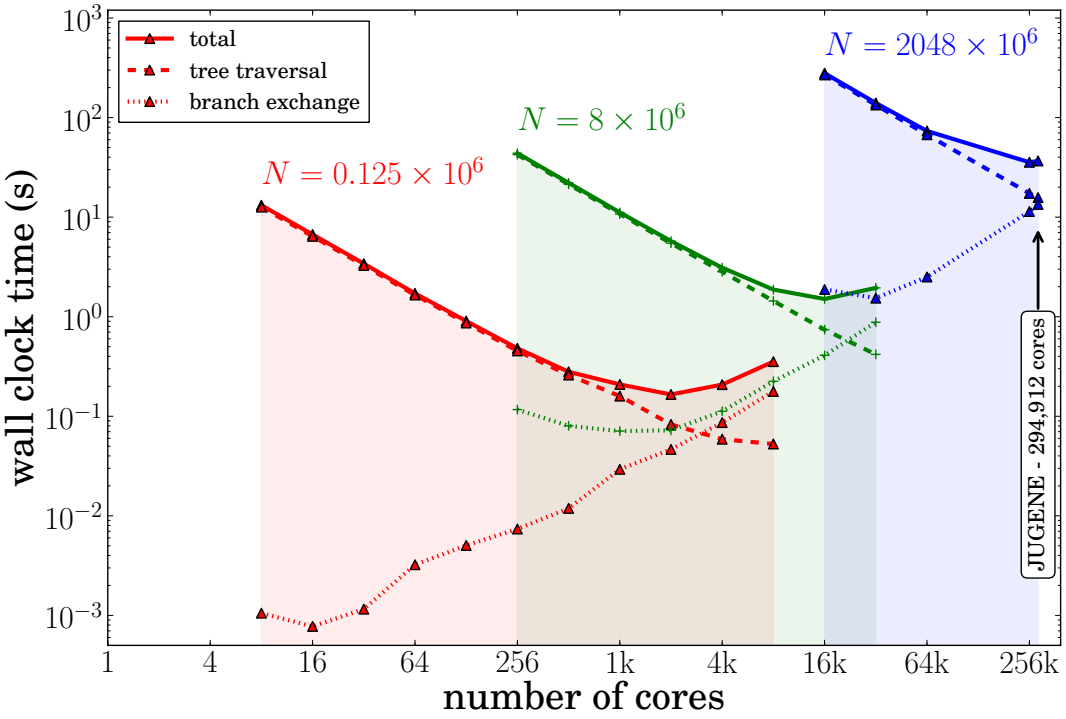


Figure 2: Comparison of the scalability of the global exchange of branch-nodes and the tree-traversal including the force-summation. Each test series simulates homogeneously distributed particles with differing numbers of particles N. [7, section 3.3]

Figure 2 shows, that up to a certain amount of cores, the tree-traversal dominates the overall run-time, although scaling perfectly even with very few particles per core. The problematic part of the current implementation is the global exchange of branch-nodes, because it scales linearly at best and thus does not scale at all.

As already mentioned in section 2, every MPI-rank determines a local non-empty set of branch-nodes from its local hashed oct-tree. These branch-nodes are exchanged with all other ranks in step 3 of the parallel Barnes-Hut algorithm. Given that every process contributes at least one branch, the number of branch-nodes and therefore also the amount of transferred data in the `MPI_AllGatherV` operation, depend linearly on the number of processes.

Furthermore, the set of remote branch-nodes has to be stored in the local hashed oct-tree which requires a considerable amount of local memory in the event of a huge number of ranks. The rightmost data point in figure 2 represents a run of PEPC on the full JUGENE that was executed with one MPI-rank on each of the 73,728 compute-nodes along with four threads for the tree-traversal. Under the assumption, that each rank contributes an average of 15 branch-nodes, the set of global branch-nodes takes approximately 197 Mb of memory, which are 9.6% of the total amount of memory on each compute-node.

However, the global exchange of data in a parallel Barnes-Hut tree-code is inevitable, because the multipole-properties of remote nodes are essential for the complete force-summation of the local particles. Figure 3 specifies three different kinds of nodes in a selected local tree after step 5 of algorithm 1. It moreover visualises the amount of nodes for each kind per tree-level as well as the overall amount

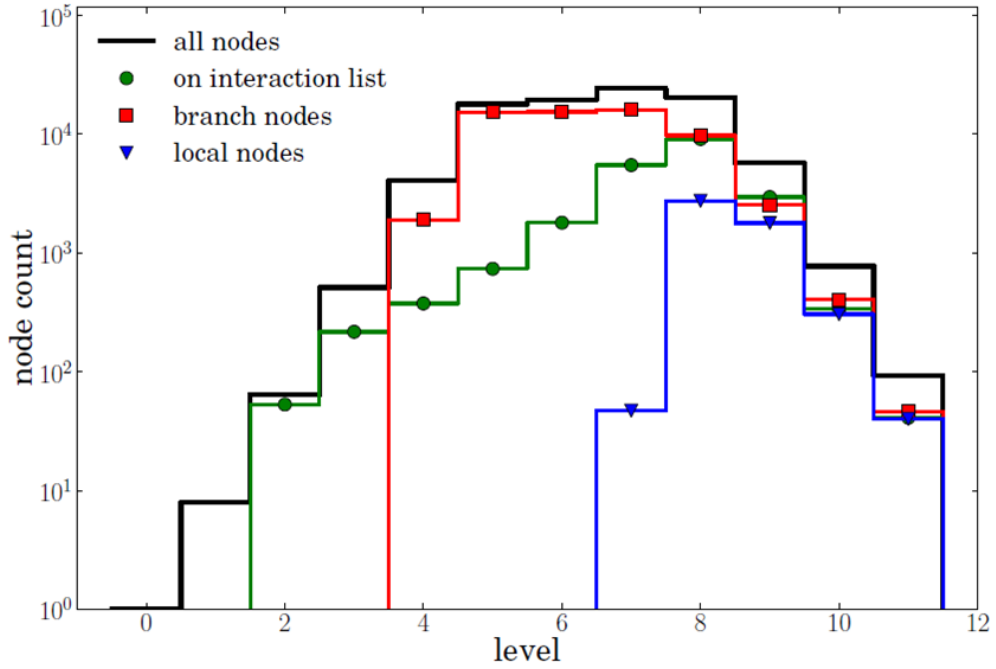


Figure 3: For selected kinds of nodes, the graph shows the amount of nodes per tree-level in a local hashed oct-tree. The most interesting aspect is the gap between the global branch-nodes and those on the interaction list in the levels 4 to 7, which implies that many branch-nodes are redundant in the local hash-table. from [1].

of nodes. The nodes, which interacted with at least one local particle, are illustrated by the line labelled "on interaction list". If another local tree had been chosen, the data points representing the branch-nodes would be the same.

Obviously, the amount of branch-nodes substantially exceeds the number of nodes on the interaction list on some levels. This implies, that a considerable amount of branch-nodes remains unused on certain levels and therefore does not really have any value for both tree-traversal and force-summation of the local particles. They have only been used to calculate the multipole-properties of the nodes between the branches and the root-node.

The above mentioned suggests, that there is the opportunity to save not only memory for the local hash-tables, but also a big chunk of data transfer during the global exchange of the branch-nodes. Furthermore, the structure of the tree above the branches, as well as the multipole-properties of the nodes in these tree-levels, are exactly the same on all ranks. They are nonetheless computed redundantly by every process during the construction of the global tree.

The problem is, that it is impossible to characterise which remote branches are dispensable before they are exchanged, because this solely depends on the structure of the remote trees, which, at this stage, is yet unknown. Unfortunately, the branch-nodes serve as entry points to the structural information during the traversal of the remote parts of the tree. Therefore, in order to reduce the amount of globally exchanged data, the coupling of the local trees has to be detached from the branch-nodes. A concept on how to accomplish this is illustrated in the next section.

4 Optimisation of the global data exchange in PEPC

Upon construction of the global tree, every rank needs the multipole-properties of the children of the root-node in order to start the tree-traversal for the local particles. In case the local availability of the multipole-properties is not ensured, their locality is required. The same consideration applies to all nodes between the root-node and the branch-nodes. To compute the multipole-property of a node, all multipole-properties of its children have to be locally available. A problem arises if these children are spread over more than one process. In this case, they have to be collected entirely by at least one rank prior to computation.

Starting with these considerations, at the very least the root-node has to be exchanged between all processes, assuming that one process already collected all essential data. The advantage is, that the amount of data received in this very last step is reduced to the absolute minimum of only one node per rank. It is important to note, that the amount of received data is independent of the number of ranks. On the downside, all remote branch-nodes have to be collected in advance by at least one MPI-process. Furthermore, the load-balancing is very bad, because all other processes are idle while the root node is computed. This constitutes a huge waste of valuable resources.

One way to keep the amount of received data per process in the global data exchange operation independent of the number of processes, is to fix the number of globally exchanged nodes. This thought, along with the desire to distribute the computational effort among more processes, leads to the idea of a single level in the oct-tree from which the nodes are globally exchanged[2].

Obviously this approach requires a certain amount of communication at a deeper tree-level, as the children of a blev-node are generally spread over more than one rank. The global tree is built in several phases, hence the strategy is termed *hierarchical tree construction*.

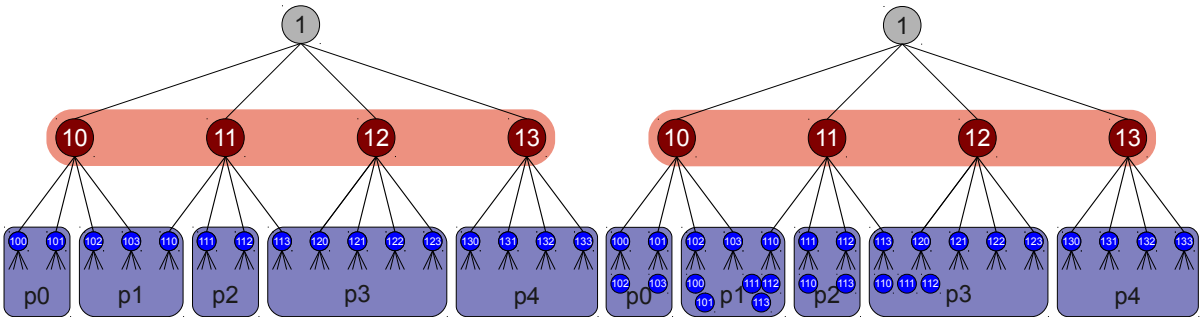
Intent on connecting the local trees to one global tree by globally exchanging the nodes in exactly one level of the tree, the following section describes the respective algorithm as well as implementational hints on how to achieve good performance with MPI. Additionally some scaling results on JUROPA are presented.

4.1 An algorithm for Hierarchical Tree Construction in PEPC

In general, more than one rank contributes to one node in the level *blev*. As already mentioned before, all ranks which contribute to a certain blev-node have to exchange data in order to calculate its multipole-properties. Since the branch-nodes are the nodes in the highest local tree-level, whose multipole-properties are already completely computed, they are predestined to be exchanged. The challenge is to ensure that only the ranks, which contribute to a blev-node, exchange the corresponding branch-nodes.

Thence, the first step in the hierarchical tree construction algorithm is to determine the unique set of contributing ranks for each blev-node. Such sets are called *groups* and the respective algorithmical step is called grouping. Every rank is member of a maximum of two groups, one for its first and one for its last blev-node, which is a consequence of sorting the particles during the domain decomposition. The

[2] As of now the single tree-level of global exchange is called *blev* and the corresponding nodes in the global tree are called *blev-nodes*.



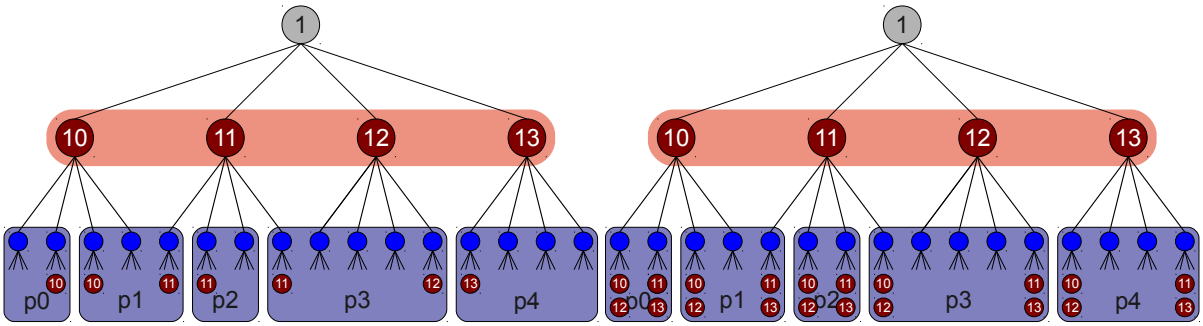
(a) initial state (b) during group-exchange

Figure 4: Example with *blev* = 1 (red bar) and 5 MPI-ranks (blue boxes). For reasons of simplicity the depicted tree is a quad-tree instead of an oct-tree and the branch-nodes are all located in level two. (a) The processes p0 and p1 contribute to the blev-node with the key 10 and therefore coalesce in a group. Process p1 is member of a second group, since it also contributes to node 11, along with p2 and p3. Ultimately, groups for the nodes 12 and 13 are not necessary, since their multipole-properties can be calculated by the processes p3 and p4 alone. The current implementation with `MPI_Comm_Split` however, creates a group with one member for blev-nodes 12 and 13. (b) The group-members exchange the branches of the corresponding blev-node.

only exceptions to that are the ranks with the highest and lowest rank number, which are member of one group at most.

There are many conceivable ways to implement grouping with peer-to-peer or one-sided communication. The current version of PEPC uses two calls of the standard MPI procedure `MPI_Comm_Split`, where the least significant four bytes of the eight byte key of the corresponding local node in the level *blev* serve as input argument for characterising the membership of the rank to the resulting MPI-communicators.

The order of the calls is not arbitrary, because all members of a group have to meet in the same call. This can be achieved by counting the local number of keys between the first and the last blev-node and



(c) before the exchange of blev-nodes (d) after the exchange of blev-nodes

Figure 4: continued example: (c) Each process builds up the tree from the available branches of all local groups up to the level *blev*. (d) Globally exchange the blev-nodes and build up the tree up to the root-node. Process p0 contributes blev-node 10 and p1 contributes blev-node 11 to the global exchange. Process p2 does not contribute any blev-node, since blev-node 11 is already contributed by p1. The blev-nodes with the keys 12 and 13 are contributed by processes p3 and p4 respectively.

grouping determine the groups according to each rank's contribution to the nodes in the level *blev*

group-exchange exchange the branches in each group and calculate the multipole-information up to the level *blev*

rooting globally exchange the nodes in the level *blev* and calculate the multipole-information up to the root node

Algorithm 2 main steps of the parallel Barnes-Hut algorithm with hierarchical tree construction

for all time steps **do**
  step 1: domain decomposition
  step 2: building up of the local tree
  step 3: hierarchical construction of the global tree
  step 3.1: grouping
  step 3.2: group-exchange
  step 3.3: rooting
  step 4: tree-traversal and force-summation
end for

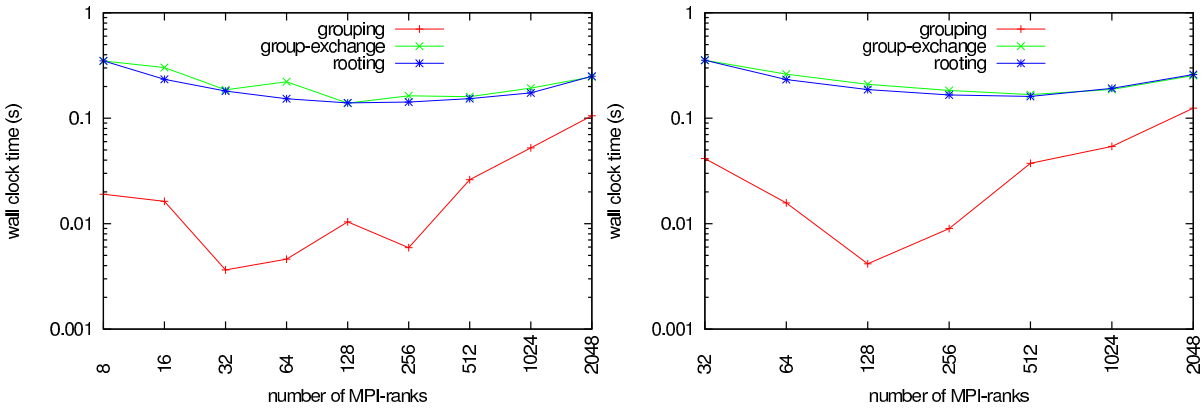
performing a prefix sum on this value with the collective `MPI_Scan` procedure. Hence all ranks with an odd $(key - prefix\_sum)$ meet in the first call and those with an even $(key - prefix\_sum)$ in the second call.

The second step encompasses the exchange of each group's branch-nodes between all group members, shown in figure 4 (b), as well as the calculation of the multipole-properties from the local branches and group-branches up to the level *blev*. In case the groups are constructed in the aforementioned way, the first part can be implemented by a call of `MPI_AllGather` and a call of `MPI_AllGatherV`. For each rank in the group, the number of nodes, that have to be exchanged, is communicated by `MPI_AllGather` to every group member. `MPI_AllGatherV` then exchanges the branch-nodes between all group members. The MPI communicator for both calls is the one from the corresponding `MPI_Comm_Split`.

The global exchange of the blev-nodes is the final step of the hierarchical tree construction. Afterwards every rank has all the information necessary to calculate the multipole-properties of the root-node required to start with the last step of the parallel Barnes-Hut algorithm: the tree-traversal and force-summation for each particle. The exchange is implemented utilising the usual `MPI_AllGather` and `MPI_AllGatherV` procedures. Every rank contributes all, at this stage locally available, blev-nodes with the exception that the blev-nodes belonging to a group are exclusively shipped by the process with the MPI-rank number zero, which belongs to the corresponding MPI communicator of this group. The main aspects of the algorithm are illustrated by the diagrams in figure 4.

The hierarchical tree construction supersedes the steps three and four (global exchange of the branches and building up of the global tree) of the original parallel Barnes-Hut algorithm, replacing them with the steps grouping, group-exchange and rooting as shown in algorithm 2.

Compared to the original algorithm, the advantage is, that every rank, in the worst case, receives all blev-nodes during the global data exchange. In the rooting step, the amount of received data per rank is thus independent of the number of MPI processes. Despite that, the average number of group members will increase with an increasing number of processes, because the number of groups is bounded above by the number of blev-nodes. This implies that the amount of received data per rank during the group-exchange still depends on the number of MPI processes.



(a) blev = 1 and 6,000,000 particles

(b) blev = 2 and 24,000,000 particles

Figure 5: Timings of the main steps of the hierarchical tree construction.

However, the amount of exchanged branch-nodes per rank during the group-exchange is much less than the amount of exchanged branch-nodes per rank in the former algorithm. Since only the branches of the local groups are exchanged, the overall amount of transferred data should be reduced significantly. Beyond that, the group-exchanges for different groups can run concurrently due to the fact that they are performed on individual MPI communicators.

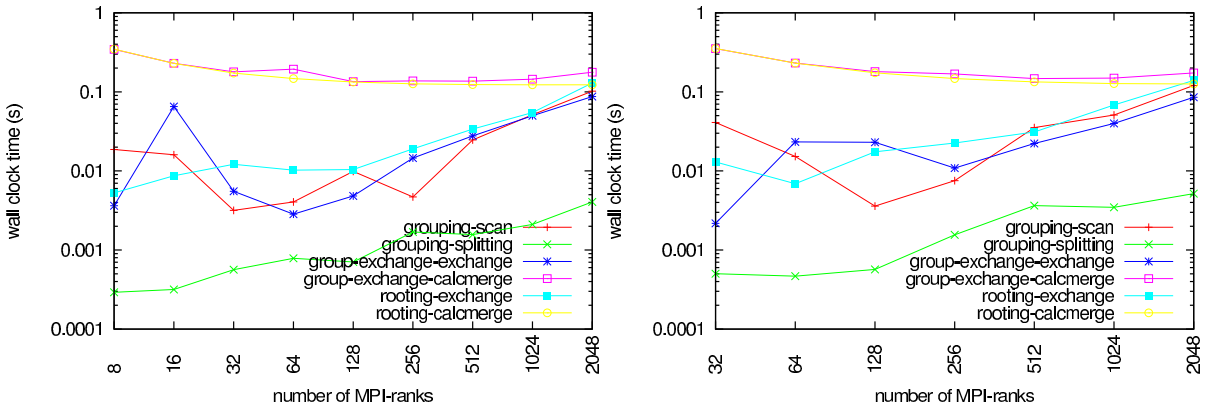
The question is, whether or not the additional collective MPI operations during grouping and group-exchange in the current implementation generate even more overhead and whether the group-exchanges for different groups are really performed concurrently. Furthermore, the hierarchical tree construction should have an impact on the run-time of the tree-traversal. In comparison to the original algorithm, where all branch-nodes are already available to all processes, the structure of the remote trees is only known down to the level *blev* prior to the tree-traversal. Processes contributing to the same blev-node constitute an obvious exception to that.

4.2 Results

The intention of this section is to discuss the scalability of the implementation of the hierarchical tree construction algoritm, presented in section 4.1, as well as to uncover other potential problems in the current implementation. Possible approaches for further improvements are mentioned in section 5.

The strong-scaling benchmarks were performed on JUROPA using 8 up to a maximum of 2,048 MPI-ranks and 4 ranks per compute-node. The test series with $blev = 1$ simulates 6,000,000 homogeneously distributed particles, which implies that the amount of particles per rank (ppr) lies between 750,000 and 2,929. The test series with $blev = 2$ simulates 24,000,000 homogeneously distributed particles with down to 11,718 ppr. Higher particle numbers and other values for *blev* have been tested as well, but are not discussed here, as they do not provide further insight.

Figure 5 shows the maximal timings over all ranks for the three main steps of the hierarchical tree construction implementation for $blev \in \{1, 2\}$. Both diagrams display similar behaviour and, on first view, good results for the steps group-exchange and rooting. The big increase in the group-



(a) blev = 1 and 6,000,000 particles

(b) blev = 2 and 24,000,000 particles

Figure 6: Timings of the sub-steps of the hierarchical tree construction.

ing step is discussed with the aid of the diagrams in figure 6, which show the sub-steps to the main steps.

It seems, that the call of the collective `MPI_Scan` procedure is the dominant part in the grouping step and that the two calls of `MPI_Comm_Split` are of minor relevance. However, figure 7 suggests that this must be questioned. It turns out, that the problem is not the poor scaling behaviour of `MPI_Scan`, but bad load balancing in the steps before the hierarchical tree construction, namely the domain decomposition and the building up of the local trees. This was confirmed by additional benchmarks with 1024 MPI-ranks and a call of `MPI_Barrier` before the call of `MPI_Scan`. With this configuration, the sub-step grouping-scan takes about 0.00125 seconds with both $blev = 1$ and $blev = 2$. Therefore the grouping step only has a minor impact on the run-time.

The steps group-exchange and rooting are dominated by the sub-step calcmerge and not by the actual exchange, at least up to 2048 MPI-ranks. Mainly, calcmerge calculates the multipole-properties

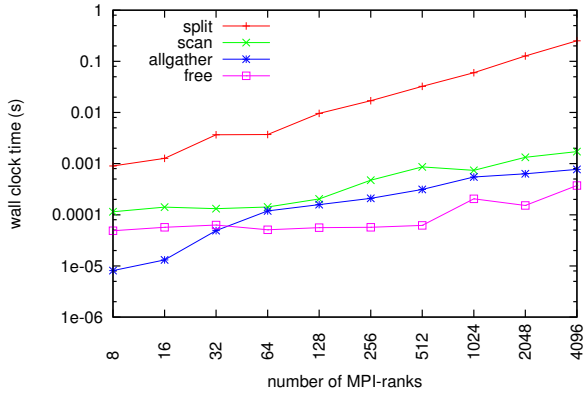
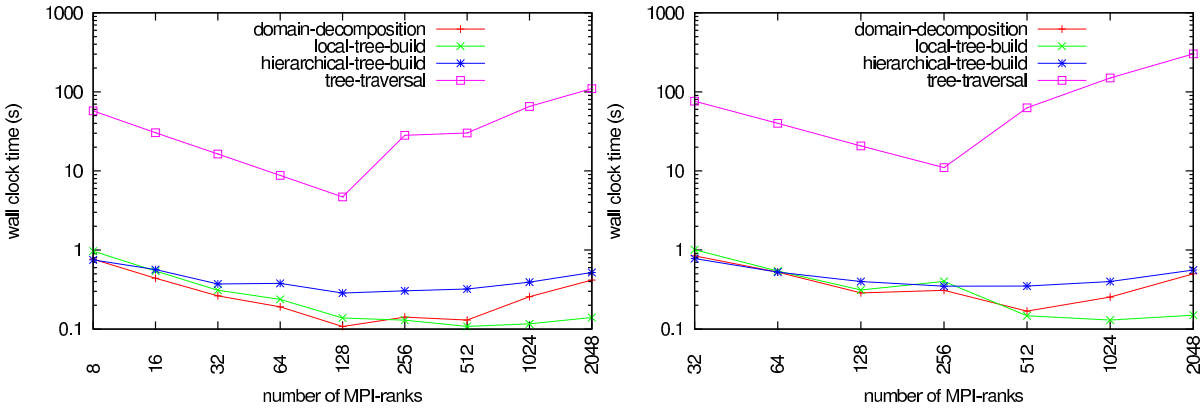


Figure 7: The scaling behaviour of selected collective MPI procedures on JUROPA is depicted above. The membership value for the call of `MPI_Comm_Split` (split) is randomly generated for each rank, in such a way that 8 MPI communicators are created, which corresponds to a run of PEPC with $blev = 1$. The so created MPI communicators are used in the calls of `MPI_AllGather` (allgather) and `MPI_Comm_Free` (free).



(a) blev = 1 and 6,000,000 particles

(b) blev = 2 and 24,000,000 particles

Figure 8: Timings of the main steps of algorithm 2 in PEPC.

of the nodes starting one level above the exchanged nodes up to the level *blev* and the root-node, respectively. As a result of that, various structural information has to be updated for every node in the hash-table. This update is redundant, as it has to be performed in both group-exchange and rooting. A future objective is to eliminate the redundancy with a new approach.

Obviously, the exchange sub-steps of group-exchange and rooting show similar scaling behaviour, with at least linear dependency on the number of MPI-ranks, which is inevitable in either case, since the number of communication partners is increasing, both per group and in total. In fact, that dependency is linear for the rooting-exchange, because the amount of exchanged data per rank is constant.

In order to achieve good scaling results for the exchange sub-step of the group-exchange as well, the only conceivable possibility is to increase the level *blev*. On the one hand, it increases the number of groups and thereby improves concurrency and on the other hand it decreases the average group size and thus the amount of exchanged data per group.

Figure 8 illustrates the timings of the main steps of the parallel Barnes-Hut tree-code PEPC with hierarchical tree construction. Up to 128 or alternatively 256 MPI-ranks, all steps scale rather well and the tree-traversal scales perfectly. Beyond 256 ranks, the tree-traversal no longer scales. It was not yet possible to uncover the reasons as to why this happens, though. It is, however, unlikely caused by the hierarchical tree construction, since tests of the original implementation on JUROPA show the same behaviour.

Incidentally, the setup of the compute-nodes was not optimal for the tree-traversal, as there was only one worker thread per MPI-rank for the hybrid tree-traversal, even though a higher number of threads would have been feasible. Therefore, the timings of this step could be reduced significantly. With three worker threads for example, the timing of the tree-traversal is reduced to approximately 18 seconds for 1,024 MPI-ranks and $blev = 2$ compared to 150 seconds with the suboptimal configuration. Naturally, this has no substantial impact on the scaling results.

5 Summary and Outlook

The aim of this paper was to describe the scalability issues in the current implementation of the parallel Barnes-Hut tree-code PEPC and to optimise the global exchange of data. This has been achieved by limiting the global data exchange to a single tree-level, which demands the design and implementation of an algorithm for the hierarchical construction of the global tree.

For the purpose of prospectively gaining deeper insight into the scalability of the new algorithm, more scaling tests are necessary. These have to be performed with an even further increasing number of ranks and gradually less particles per rank. To that end, tests on JUGENE are aspired to, as well. Prior to that, however, the source of the strange scaling behaviour of the tree-traversal has to be uncovered and eliminated.

Additional potential for improvement lies in the optimisation of the memory usage in the hash-table. Pending some deliberation, the amount of reserved entries for global branches can be reduced, thus making more space for particles available.

The single level of global data exchange in the current implementation is a predefined value and constant over all time-steps and ranks. It is almost freely selectable in the range of $[1, 10]$. The upper bound originates from the data type of the color-argument of the MPI procedure `MPI_Comm_Split`, which is only a 4 byte integer. This limitation is not really noteworthy, since such a high level of global data exchange is not practical at all, as the amount of nodes, that would have to be exchanged, is ridiculously high.

An error regularly occurs in case an inhomogeneous particle distribution is simulated with $blev > 2$, because several branch-nodes are usually located in the tree-levels above the level *blev*. This problem can easily be avoided. Instead of the branch-nodes, one exchanges the corresponding nodes in the level *blev* and a higher level respectively, if no blev-node exists. Automatically determining an "optimal" value for *blev* at the beginning of each time step might also be expedient in terms of improving simplicity and user friendliness.

Acknowledgements

I gratefully acknowledge the Jülich Supercomputing Centre and the JSC-staff, especially Lukas Arnold and Mathias Winkel for supervising my project, Natalie Schröder and Mathias Winkel for the organisation of the Guest-Student Programme, Paul Gibbon for PEPC, Florian Janetzko, Brian Wylie and Peter Philippen for their help, advice and patience, Claudia and Michael Knobloch for the invitations to Jülich and Aachen and Wolfgang Walter for recommendation. Last but not least, many thanks go to all the other guest students for the great time.

References

1. L. ARNOLD. private communication.
2. J. BARNES, J. AND P. A. HUT. *A hierarchical $O(NlogN)$ force-calculation algorithm*, *Nature*, 324:446-449, 1986.

3. P. GIBBON, R. SPECK, L. ARNOLD, M. WINKEL, H. HÜBNER. *Parallel Tree Code*, *Proceedings of the WE-Heraeus Summer School 2010 on Fast Methods for Long-Range Interactions in Complex Systems*, Forschungszentrum Jülich, 65-84, 2010
4. H. HÜBNER. *A Priori Minimisation of Algorithmic Bottlenecks in the Parallel Tree Code PEPC*, *Diploma thesis*, Forschungszentrum Jülich, July 2011
5. S. PFALZNER AND P. GIBBON. *Many Body Tree Methods in Physics*, *Cambridge University Press*, New York, 2005
6. M. S. WARREN AND J. K. SALMON. *A parallel hashed Oct-Tree N-body algorithm*, *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, Portland, Oregon, United States, 12-21, 1993
7. M. WINKEL, R. SPECK, H. HÜBNER, L. ARNOLD, R. KRAUSE, P. GIBBON. *A massively parallel, multi-disciplinary Barnes-Hut tree code for extreme-scale N-body simulations*, submitted to *Computer Physics Communications*

How fast are local Metropolis updates for the Ising model on a graphics card

Momchil Ivanov

Universität Leipzig
Fakultät für Physik und Geowissenschaften
Linnéstraße 5

E-mail: momchil@xaxo.eu

Abstract:
This report gives implementation details and results from a computer program that has been created for simulating the Ising model with local Metropolis updates on a present-day NVIDIA GPU architecture for scientific computing. The results are comparable with implementations of a similar model on the same hardware architecture from [1]. Correctness of the code is illustrated by providing results of physical observables from conducted simulations using the program. Speedup results with regard to a CPU implementation of the algorithm are provided for different system sizes and ECC enabled/disabled GPU memory. Short discussion on the implementations of the random number generators for the GPU that have been used (Mersenne Twister and XORWOW) is provided together with performance comparison, since the time cost is on the order of the Metropolis updates.

1 Introduction

Graphics Processing Units (GPUs) have been used for solving computationally intensive tasks for the past few years. Nowadays these devices labelled "general purpose" find application in the field of high performance super computing. This work presents an implementation of a Monte Carlo simulation of the Ising model using a local Metropolis update for one such device family, namely the NVIDIA Tesla M2050/M2070, based on an implementation from [1]. Different techniques for optimising the performance of the algorithm are discussed and the obtained results are compared with those from [1] for a similar system. Measurement times for the generation of random numbers and the measurement of observables are also given, since this is part of the whole simulation process. However, they are not optimised as that is not in the scope of this report. Some problems that occurred when working with CUDA are given at the end of the report.

2 Theoretical background

2.1 The Ising model

The Ising model is a simple model of a magnet. The model consists of discrete objects called spins which can take one of two different states: $\sigma_i \in \{-1, 1\}$. The spins are placed on a lattice or graph and each spin interacts only with its nearest neighbours. The energy E of the system is defined as:

$$E = -\sum_{<ij>} \sigma_i J_{ij} \sigma_j - \sum_i h_i \sigma_i \tag{1}$$

where the sum is over all pairs of nearest neighbours $< ij >$. The coupling constants must fulfil $J_{ij} = J_{ji}$ and one recognises 2 distinguished cases:

- J_{ij} = const, the isotropic case known as the Ising model
- $J_{ij} \in \{-1, 1\}$, with $h_i = 0$ known as the Edwards-Anderson spin glass model.

The following work considers mainly the Ising model without external magnetic field, i.e. $J_{ij} = J$ and $h_i = 0$. The implementation uses a 3-dimensional simple cubic lattice with periodic boundary conditions illustrated on fig. 1.

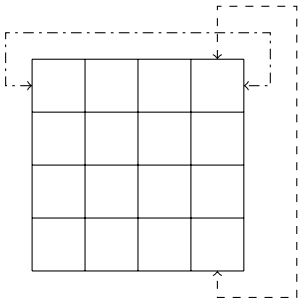


Figure 1: Toroidal periodic boundary conditions (usually referred to as periodic boundary conditions) on a 2-dimensional lattice.

2.2 Observables

The partition function Z of the system and the expected value of an observable (thermodynamic quantity of interest) $< O >$ are defined as:

$$Z = \sum_\sigma e^{-\beta E(\sigma)} \tag{2}$$

$$< O > = \frac{1}{Z} \sum_\sigma O(\sigma) e^{-\beta E(\sigma)} \tag{3}$$

where the sum is over all spin configurations σ; $\beta = \frac{1}{k_B T}$ is the inverse temperature, with k_B the Boltzmann constant and T the temperature. The units of β in this work are chosen such that $k_B = 1$ and $J = 1$. The basic observables used to check the correctness of the implementation of the simulation program are the internal energy E and the magnetisation of the system $M = \sum_i \sigma_i$.

2.3 Solution

Exact solutions of the Ising model exist for 1 dimension [2] and 2 dimensions [3] without external magnetic field, i.e. $h_i = 0$. The 3-dimensional Ising model has not been solved yet. There are 2^{L^3} states in 3D, which makes exact enumeration of large systems impossible, therefore the well known Monte Carlo simulation method is being used to sample the state space and estimate the expected values of observables.

2.4 The Metropolis Update Algorithm

The Monte Carlo simulation is implemented using the Metropolis update algorithm. In it's original form one needs 2 random numbers per spin update which is highly inefficient. One can use 1 random number per spin update which results in a more efficient and common implementation, see alg. 1. The latter has been used in this work.

Algorithm 1 Metropolis update algorithm, efficient form.

1: **for** $i := 1$ to N **do**
2: calculate ΔE for flipping the spin σ_i
3: flip the spin with probability $\min\{1, e^{-\beta \Delta E}\}$
4: **end for**

N is the number of spins in the system, which is equal to the volume, hence in 3 dimensions $N = V = L^3$. The energy difference ΔE is defined as:

$$\Delta E = 2\sigma_i \sum_{n}^{neighbours} \sigma_{j_n}. \tag{4}$$

3 Going parallel: vectorisation

3.1 Modern tools: GPU

The main goal of using vectorisation is the speedup from updating multiple spins at the same time. This has been done long time ago on vector machines [4]. The goal of this work is to create an efficient implementation for the present-day GPU hardware architecture of NVIDIA. The main advantages of these cards are: large number of computing units, small physical dimensions and less power consumption per computing unit than a CPU core.

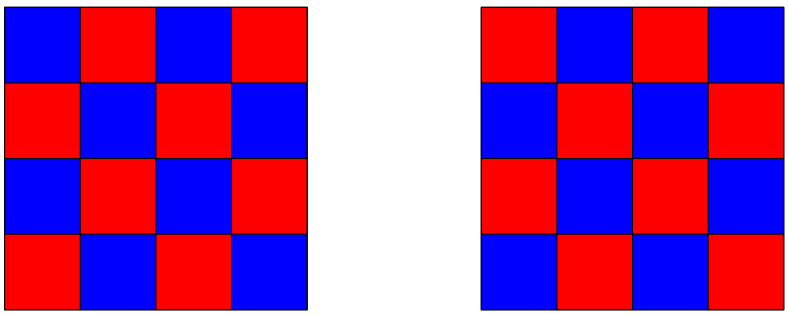
For example the NVIDIA GPU Tesla M2070 @ $1,15$ GHz consumes $\sim 0,5$ W/ core [5], compared to Intel(R) Xeon(R) CPU X5650 @ $2,67$ GHz $\sim 15,8$ W/ core [6]. Considering the clock frequency the difference is about a factor of 10, which makes GPU clusters cheaper to operate than CPU clusters delivering the same performance. The Jülich Supercomputing Centre operates currently one computer cluster (**judge**) based on NVIDIA Tesla M2050 cards with ECC switched on and a smaller test cluster (**mini-judge**) equipped with NVIDIA Tesla M2070 with ECC switched off. Both GPUs have the same computational power and offer 448 computing units (CUDA cores) as one can see from the technical specifications in table 1. The only difference between both models is the size of the main memory (Global memory). The present work is based on these cards, hence the data structures and algorithms being used are chosen such as to make efficient use of the hardware. The code exploits the presence of L1 and L2 memory for caching access to Global memory, hence no manual caching using Shared memory is needed. Moreover, this makes program code more readable.

Compute capability	2.0
Multiprocessors	14
Warp size	32
CUDA Cores	448
Global memory	3/6 GB
L2 cache size	786432 B
GPU Clock Speed	1,15 GHz
Registers per block	32768
Threads per block	1024

Table 1: Technical specifications of NVIDIA Tesla M2050/M2070.

3.2 Checker board decomposition

The first step of vectorisation is the lattice decomposition in 2 sets (red, blue) of non interacting spins illustrated on fig. 2. This restricts the lattice length to a multiple of 2, i.e. $L = 2k, k \in \mathbb{N}$. The maximum speedup that one can achieve using this technique is $\frac{N}{2}$. For the $L = 16$ lattice, the maximum speedup would be 2048.

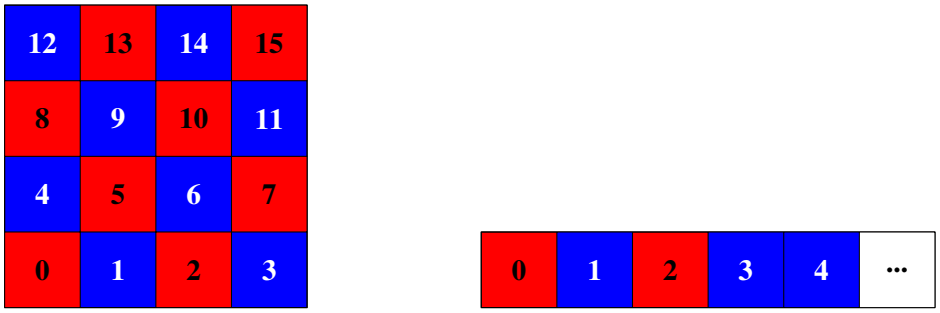


(a) z coordinate has even value (b) z coordinate has odd value

Figure 2: Checker board decomposition: xy planes for even and odd values of the z coordinate.

3.3 Lattice representation in memory

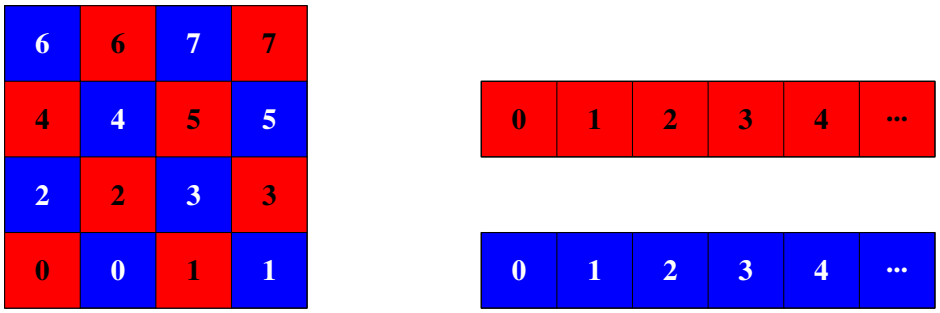
The most straight forward way to represent a 3-dimensional lattice is as a 1-dimensional array with the mapping $(x, y, z) \rightarrow i : i = x + yL + zL^2$. In computer memory this leads to a structure (fig. 3) which does not produce a good memory access pattern on the GPU when calculating ΔE. When updating the set of red spins every thread on the GPU first will load the term σ_i which results in access to spins 0, 2, 5, 7, ... and in this case the access needs to be serialised. In order to achieve maximum performance on NVIDIA devices one needs to have coalesced memory access, which means that the red spins need to lay sequentially in the memory. The same applies in the case when one is trying to update the set of blue spins. This can be achieved by storing both sets in different 1-dimensional vectors, illustrated on fig. 4.



(a) xy plane of a 3-dimensional array, z = 0

(b) 1-dimensional array

Figure 3: Representation of a 3-dimensional lattice in an 1-dimensional array, where each number indicates the position of that lattice point in the 1-dimensional array.



(a) xy plane of a 3-dimensional array, z = 0

(b) two 1-dimensional arrays

Figure 4: Representation of a 3-dimensional lattice in two 1-dimensional arrays, where each number indicates the position of that lattice point in the corresponding 1-dimensional array.

Another way of achieving better performance could be to coalesce the access to the neighbouring spins σ_{j_n} but is not discussed in this work, since it would be a considerable effort. Moreover, it is not clear if it will provide performance increase.

3.4 Coordinate mapping and calculation of neighbours

The representation of the 3-dimensional lattice in a 1-dimensional array requires a different way for computing indices of neighbouring spins. As one sees from fig. 4 the indices of the neighbours on the y and z axes do not depend on the colour of the spin. The indices of these neighbours can be computed in the following way:

$$z_+ = (i + \frac{L^2}{2}) \mod \frac{L^3}{2} \tag{5}$$

$$z_- = (i - \frac{L^2}{2} + \frac{L^3}{2}) \mod \frac{L^3}{2} \tag{6}$$

$$y_+ = (i - i \mod \frac{L^2}{2}) + (i + \frac{L}{2}) \mod \frac{L^2}{2} \tag{7}$$

$$y_- = (i - i \mod \frac{L^2}{2}) + (i - \frac{L}{2} + \frac{L^2}{2}) \mod \frac{L^2}{2} \tag{8}$$

where z_+ is the neighbour in the positive z direction, z_- is the neighbour in the negative z direction, y_+ and y_- respectively. On the x axis one needs to take care of the values of the y and z coordinates and the colour of the spin. The neighbours can be computed in the following way:

$$z = i \div \frac{L^2}{2} \tag{9}$$

$$y = (i - z\frac{L^2}{2}) \div \frac{L}{2} \tag{10}$$

$$d = (z \wedge 1) \oplus (y \wedge 1). \tag{11}$$

For a red spin:

$$x_+ = (i - i \mod \frac{L}{2}) + (i + 1 - d \oplus 1) \mod \frac{L}{2} \tag{12}$$

$$x_- = (i - i \mod \frac{L}{2}) + (i - 1 + d + \frac{L}{2}) \mod \frac{L}{2}. \tag{13}$$

For a blue spin:

$$x_+ = (i - i \mod \frac{L}{2}) + (i + 1 - d) \mod \frac{L}{2} \tag{14}$$

$$x_- = (i - i \mod \frac{L}{2}) + (i - 1 + d \oplus 1 + \frac{L}{2}) \mod \frac{L}{2}. \tag{15}$$

Here $\oplus$ is bit wise **XOR**, $\wedge$ is bit wise **AND** and $\div$ is integer division **div**. In order to achieve best performance the computer code uses bitwise shift operations with the following identities for $b = 2^k, k \in \mathbb{N}$: $a \div b = a >> \log_2 b$ and $a \mod b = a \wedge (b-1)$. This restricts L to being a power of 2, hence $L = 2^k, k \in \mathbb{N}$. I acknowledge the use of ideas and computer code for the neighbour

computation from the computer code accompanying [1], moreover I have optimised their formulas for computing neighbours on the x axis.

3.5 Algorithm

The algorithm used is composed of different phases: generation of random numbers, Metropolis update of all spins and a measurement phase. The order of the phases is given in alg. 2. The random numbers are generated for the whole lattice, before the Metropolis routines are executed, in order to avoid branching of the GPU code. The latter is supposed to deliver maximum performance.

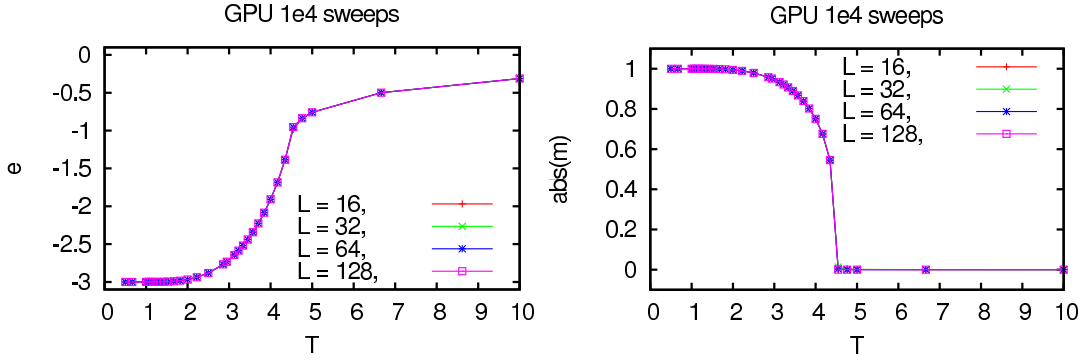
Algorithm 2 Simulation algorithm.

1: **for all** simulation steps **do**
2: generate random numbers
3: update all red
4: update all blue
5: measure every n loops
6: **end for**

4 Results

4.1 Physical results

In order to check the correctness of the simulation program, results for the values of the internal energy and magnetisation have been obtained at different temperatures. The behaviour of both curves seems to be correct as one can see on fig. 5. A phase transition is expected at a value of $\beta = 0.22165703$, compare [7].



(a) Internal energy e per spin ($e = \frac{E}{N}$)

(b) Magnetisation m per spin ($m = \frac{M}{N}$)

Figure 5: Observables in dependence of the temperature T for systems of size $L = 16, 32, 64, 128$.

4.2 Update times

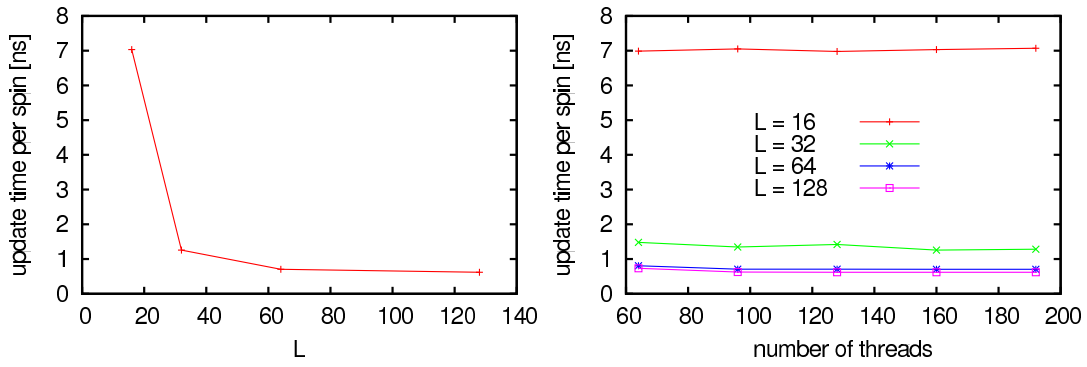
Simulations with 1000 sweeps have been done in order to estimate the Metropolis update time per sweep for the whole lattice. Dividing that by the lattice volume, one gets the Metropolis update time per spin given in table 2. Estimates of the time needed for the different phases are given in table 3. Measurements of observables have been done partly on the CPU and there is an extra time cost for copying the data from the GPU memory to the CPU memory. This is highly inefficient and could be optimised with direct computation on the GPU, but is not in the scope of this work. The update times have been compared to results for a similar system with $L = 128$ from [1]. They quote Metropolis update time per spin 0.66 ns on NVIDIA GTX 480 card, compared with 0.62 ns I get with 160 threads per block on the NVIDIA Tesla M2070 card. The update times do not depend much on the number of threads per block used, as one can see on fig. 6, nevertheless the fastests results have been obtained with 160 threads per block.

L	t in ns	t_{ECC} in ns
16	7,03	15,47
32	1,26	2,50
64	0,71	0,85
128	0,62	0,63

Table 2: Time for one Metropolis update for different lattice sizes with ECC memory function switched off (t) and on (t_{ECC}).

Phase	t in µs
Metropolis	1296
RNG	252
copy data to host	5290
measure E	3825
measure M	472

Table 3: Time decomposition in different phases for one iteration on a lattice with $L = 128$: Metropolis update, generation of random numbers, memory copy GPU $\rightarrow$ CPU, measurements of E and M.



(a) in dependence of the system size, 160 threads per block

(b) in dependence of the number of threads per block

Figure 6: Scaling of the metropolis update time (in ns).

Note that in order to do scientific research, one needs reproducible results, which means that the use of ECC memory is a must. For test runs one might resort to the use of non ECC memory in order to get results faster.

4.3 Speedup

In order the estimate the speedup from using the GPU instead of the CPU, a test run has been done with a CPU program using the same algorithm on 1 core of Intel(R) Xeon(R) CPU X5650 @ $2,67$ GHz processor. The program has been compiled with Intel compiler 11.1.072 (optimisation level -O3). As one sees from the results given in table 4, there is no speedup when using the GPU with ECC memory for small systems. This is due to the system size, which is not able to utilise the full computing power of the device because of the latency of the memory access. For larger systems the speedup is almost independent of the ECC memory, but these systems need a lot more simulation steps in order to get decent results.

L	t_{CPU}	t_{GPU}	$t_{GPU,ECC}$	t_{CPU}/t_{GPU}	$t_{CPU}/t_{GPU,ECC}$
16	12	7,03	15,47	1,7	0,7
32	11	1,26	2,50	8,7	1,2
64	11	0,71	0,85	15,4	12,9
128	10	0,62	0,63	16,1	15,8

Table 4: Metropolis update time per spin in ns for a CPU and a GPU program using the same algorithm.

4.4 CUDA kernel profiling data

The profiling data has been obtained with the **computeprof** profiler from NVIDIA for simulations of 1000 sweeps on lattices of different sizes, listed in table 5. The data shows that the computing kernels achieve almost maximum performance (IPC) on the largest system, which indicates that the memory latency is minimal and most memory access is done via the L1 and L2 caches.

	$L = 16$	$L = 128$
Limiting factor:		
Achieved Occupancy (theoretical 0,83)	0,2	0,8
IPC (maximum 2)	1,0	1,9
Memory Throughput Analysis:		
Kernel requested global memory read throughput(GB/s)	11,81	60,00
L1 cache global hit ratio (%)	49,09	35,94
Occupancy Analysis:		
Grid size	[13 1 1]	[6554 1 1]
Block size: [160 1 1]		
Register Ratio (16 registers per thread): 0,75 (24576 / 32768)		

Table 5: Profiling data from **computeprof** from runs with 1000 sweeps on systems with $L = 16, 128$.

4.5 Random number generators

Different RNGs have been tested for the simulations in order to check physical results and benchmark the algorithms. The Mersenne Twister implementation from [1] is rather slow with about 300 µs per

sweep on a $L = 16$ lattice, which is probably due to the large state vector. Moreover, there were some problems with the results, which might be due to a bug in my code. The XORWOW [8] implementation from the library curandlib [9] has been tested using the host API. It produces consistent results and is quite fast with about 40 µs per sweep. There are some serious disadvantages, though: no source code available for the host API, the documentation does not explain how the sequences are seeded, there is no way of direct seeding, the available device code gives some hints how seeding might be done and NVIDIA does not want to publish the source code (private communication). These facts render the host API useless for serious scientific research and can only be used for test purposes, due to the implementation being rather fast.

4.6 Portability of the computer code to other models

The program written for the Ising model has also been extended with small modifications to the Edwards-Anderson spin glass model. Results from that model are not in the scope of this report and hence not provided. The computer code can also easily be ported to all other 3-dimensional lattice models using a cubic lattice with nearest neighbours interactions.

4.7 Problems with CUDA

Since GPU hardware is still changing, there is still not a fixed set of features that every GPU should support. Considering NVIDIA GPUs, there is no guarantee that fast code for the present architecture will be still fast on the next one. Therefore the algorithms and data structures used have to be tuned for the specific hardware, hence the GPU code is not "portable" across different architecture generations. This means that these devices are not that "general purpose", though they might be programmable in a dialect of the C programming language. Moreover the documentation of CUDA [10] mixes C and C++ and does not make it clear how to use pure C and which versions are supported. Note that the "-x" flag for the nvcc compiler is not even documented in [11]. Another example of the immaturity of the CUDA documentation is that it is spread all over the website of NVIDIA and one finds different bits and pieces in different documents.

5 Conclusion

A computer program has been successfully created for simulations of the Ising model with a local Metropolis update on a GPU from NVIDIA. Examples illustrating the correctness of the implementation have been discussed together with performance details on the algorithms used. The program code written has been well optimised for the chosen architecture. The latter is proven by the profiling data and the comparison of the update times with results from others simulating a similar model. The choice of a random number generator shows a significant impact on the performance due to the big memory latency and the small cache size of present GPUs. The speedup that one gets in comparison with a CPU version of the programm is significant only for large systems, which are not always feasible for simulations. Because the GPU hardware is still changing significantly, the software is still changing too and therefore its documentation is still immature. This forces one to choose specific algorithms and data structures depending on the device gneration one wants to use. These facts render present GPU devices

not that “general purpose” as advertised. Nevertheless they deliver significat performance speedup for highly vectorisable problems with small memory footprint.

References

1. M. Bernaschi, G. Parisi, and L. Parisi. The heisenberg spin glass model on gpu: myths and actual facts. *arXiv:1006.2566v1 [cond-mat.dis-nn]*, 2010.
2. Ernst Ising. *Beitrag zur Theorie des Ferro- und Paramagnetismus*. PhD thesis, Mathematisch-Naturwissenschaftliche Fakultät der Hamburgischen Universität, 1924.
3. Lars Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Phys. Rev.*, 65(3-4):117–149, Feb 1944.
4. David Landau. Vectorisation of monte carlo programs for lattice models using supercomputers. In Kurt Binder, editor, *The Monte Carlo Method in Condensed Matter Physics*, volume 71 of *Topics in Applied Physics*, pages 23–51. Springer Berlin / Heidelberg, 1995. 10.1007/3-540-60174-0_2.
5. NVIDIA. Tesla m2050 / m2070 gpu computing module supercomputing at 1/10th the cost. `http://www.nvidia.com/docs/IO/43395/NV_DS_Tesla_M2050_M2070_Apr10_LowRes.pdf`, 4 2010. Accessed on 06.10.2011.
6. Intel. Intel® xeon® processor x5650 (12m cache, 2.66 ghz, 6.40 gt/s intel® qpi). `http://ark.intel.com/products/47922/Intel-Xeon-Processor-X5650-(12M-Cache-2_66-GHz-6_40-GTs-Intel-QPI)`. Accessed on 06.10.2011.
7. Martin Weigel and Wolfhard Janke. Error estimation and reduction with cross correlations. *Phys. Rev. E*, 81(6):066701, Jun 2010.
8. George Marsaglia. Xorshift rngs. *Journal of Statistical Software*, 8(14):1–6, 7 2003.
9. NVIDIA. *CUDA Toolkit 4.0 CURAND Guide*. NVIDIA, pg-05328-040_v01 edition, 01 2011.
10. NVIDIA. *CUDA C Programming Guide Version 4.0*. NVIDIA, 5/6/2011 edition, 6 2011.
11. NVIDIA. *The CUDA Compiler Driver NVCC*. NVIDIA, 01-27-2011 edition, 1 2011.

Volume Visualisation using the Tetrahedron Method

Kaustubh Bhat

German Research School for Simulation Sciences
Wilhelm-Johnen-Straße
52428 Jülich

E-mail: k.bhat@grs-sim.de

Abstract:
In solid state physics applications, there is a frequent need to visualise volumetric data and calculate integrals over these volumes to be able to interpret and understand the physical aspects of the model under consideration. Keeping this in mind, we develop an algorithm for finding iso-surfaces using discrete scalar data on three dimensional meshes. Once the iso-surfaces are formed, we calculate the integrals of the scalar function over the volume enclosed by the iso-surface. These volume integrals can be used to calculate information such as the electronic charge, spin etc. over the volume. The method that forms the basis of this technique is the scheme of marching tetrahedra. The method is then implemented in code so that it is possible to interactively visualise the resulting iso-surfaces. Another aspect of the code is the ability to iteratively find the iso-value, given the integral over the volume enclosed by the iso-surface.

---

1 Introduction

In physics, one often faces the need to visualise data to shed more light on the physical aspects of a particular model. This need is even bigger when the data is obtained from simulations and no direct observations are available. For example, simulations calculate he amplitude of the Wannier functions of crystal data on a rectilinear grid. Going further, it may be necessary to find, say, the electronic charge in the said crystal. This quantity is merely an integral of the modulus squared of the wave function over the crystal volume. But, as is mostly the case, wave function data is not analytically obtainable and one has to rely on approximate techniques to evaluate such quantities. Usually, the data thus obtained is discrete, and we need to come up with non-analytic methods to find these integrals. These ideas form the basic motivation in coming up with a technique to both visualise such discrete data, and simultaneously evaluate properties that are dependent on this data.

2 Method of Marching Cubes

2.1 Voxel

We begin the discussion of the method of marching cubes by introducing the concept of voxels. In physics, one has to deal with crystals which possess translational symmetry. Thus, it becomes very easy to super-impose three dimensional rectilinear meshes onto a unit cell or onto a super-cell. Such a structured mesh can be looked at as a collection of volume elements, or "voxels", which are parallelepiped entities. Most often, we have to represent some sort of scalar information on this crystal. Since we usually have only discrete data, we need a method to somehow estimate the function over the entire volume and then extract meaningful conclusions from the data.

2.2 Methodology

Iso-surface plots are one of the many ways of representing scalar information on a three dimensional mesh. The method of marching cubes is one of the most popular methods of locating an iso-surface using discrete data. The method approximates the scalar data as a linear function along the edges of the voxel. The name marching "cubes" may be misleading, as the method can be extended to any parallelepiped shaped voxel. Depending on the iso-value, we assign a '+' $(f_{ijk} \geq isoValue)$ or a '-' $(f_{ijk} < isoValue)$ to each vertex.

For some edge where we have different signs at the two vertices, we say that we have a sign change along the edge. Now we introduce the approximation that the scalar function is linear along the edge of the voxel. It is easy to deduce that in this approximation, there must be a point somewhere on the edge where the iso-value is exactly equal to the function value at that point. Let us label the vertices to be 1(+) and 2(-) respectively and let the iso-value be f, then the location of this point is simply given by

$$\boldsymbol{r} = \boldsymbol{r_1} + \frac{(f - f_1)}{(f_1 - f_2)} \cdot (\boldsymbol{r_1} - \boldsymbol{r_2}) \tag{1}$$

This point lies on the iso-surface. Using this strategy, we can locate all the iso-points on the 12 edges of a voxel. The actual iso-surface is approximated by drawing planes that pass through all the iso-points. Repeating this process on all voxels, we can join all the resultant surface patches to produce the complete iso-surface. An illustration of this process is shown in the Figure 1. If more than three edges contain iso-points, then the surface is approximated by triangular patches, determined by some suitable ordering of points.

This method is quite simple to implement. But there are various situations in which the resultant surface is ambiguous. Imagine a case as shown in Figure 2 where *two* vertices of a voxel have a different sign from the rest. In this case, we have six edges containing an iso-point. Thus, we can have different ways of creating the iso-surface patches as shown. Thus, we need an additional step to resolve this ambiguity. This can be done by choosing the patches that do not intersect the shortest body diagonal, as the probability of the occurrence of a double sign-change along this diagonal is minimum. Many

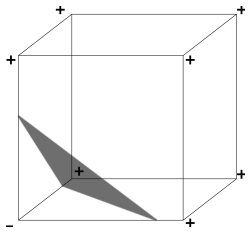


Figure 1: Iso-surface patches

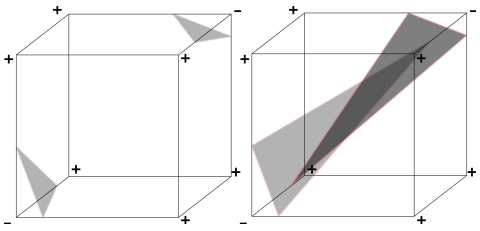


Figure 2: Marching Cubes - Ambiguity

other methods such as the midpoint decider or the asymptotic decider can also be used to resolve this ambiguity. Considering all the possible combinations of '+' and '-' vertices, since the voxel has eight vertices, we have a total of $2^8 = 256$ different cases that need to be handled. This obviously adds to the complexity of the solution.

$$\begin{aligned} f_{xyz} = {} & f_{000}\,(1-x)\,(1-y)\,(1-z) + f_{100}x\,(1-y)\,(1-z) + f_{010}\,(1-x)\,y\,(1-z) \\ & + f_{001}\,(1-x)\,(1-y)\,z + f_{110}xy\,(1-z) + f_{101}x\,(1-y)\,z + f_{011}\,(1-x)\,yz \\ & + f_{111}xyz; \quad 0 \leq x, y, z \leq 1 \end{aligned} \tag{2}$$

Another conspicuous drawback stems from the trilinear interpolation of scalar values inside the voxel. The trilinear interpolation process is described by the equation 2. The interpolation, which is difficult to visualise in three dimensions, can be contracted to just two dimensions and plotted as shown in Figure 3. The figure shows that the interpolation produces a curved surface. In comparison, when we draw an iso-surface patch in the voxel, we approximate the surface using a planar patch. This is obviously not quite consistent.

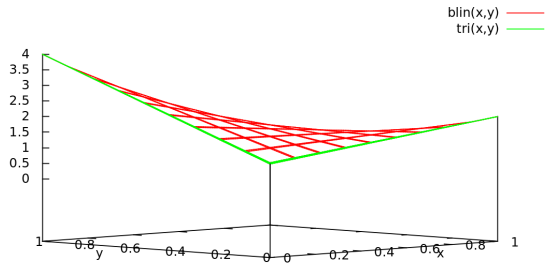


Figure 3: Marching cubes - Trilinear Interpolation vs. Planar iso-surface patch

Keeping these aspects in view, we see the need for a better method to find the iso-surface using the given data. We shall investigate such a method in the next section.

3 Method of Marching Tetrahedra

The method of marching tetrahedra can be looked at as a slight modification of the marching cubes method, where, instead of a voxel, each element is a tetrahedron. In this section we will discuss the mathematical background of the marching tetrahedra scheme followed by a discussion on how we can transition to a tetrahedral mesh, starting from the voxel mesh used for the marching cubes.

3.1 Tetrahedron

A tetrahedron is the simplest three-dimensional polyhedron, consisting of four vertices and six edges. The tetrahedra we use are convex and thus have an Euler characteristic of 2. One can immediately see that owing to the lesser number of vertices, we will have lesser cases to handle as compared to the marching cubes method. The biggest benefit we derive from using tetrahedra as our elementary volumes is that is the possibility of defining a unique linear interpolation on the simplex.

3.2 Dividing a voxel into tetrahedra

As discussed above, we already have a rectilinear mesh of voxels. It would be very beneficial to use the existing data to form the new tetrahedral mesh that we require. To this end, we would like to divide each voxel into tetrahedra. There are various approaches to achieving this end. The essential distinguishing criterion is the number of external faces shared by the tetrahedra formed by adjacent voxels. Two tetrahedra sharing a face differ in exactly one vertex. Each voxel has 6 outside faces and 8 vertices.

3.2.1 Five Tetrahedra with central tetrahedron

Here we first create a central tetrahedron that does not have any external faces and has only internal faces. Then we construct four more tetrahedra from the remaining vertices such that each one has three external faces (Figure 4). The main drawback of this approach is that the tetrahedra do not have equal volumes. Also, adjoining tetrahedra from adjacent voxels do not share a common face i.e. are misaligned.

3.2.2 Six Tetrahedra with non uniform shape

In this method we form tetrahedra of equal volume, such that three of them have three internal faces and the other three have three external faces (Figure 5). Here, although the six tetrahedra have equal volumes, the problem of misalignment of cells belonging to adjacent tetrahedra persists as the neighbouring tetrahedra do not share a common face.

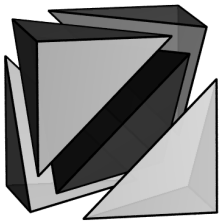


Figure 4: Five tetrahedra with a central tetrahedron

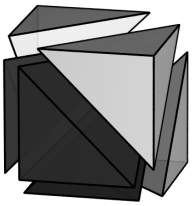


Figure 5: Non-uniform tetrahedra of equal volume

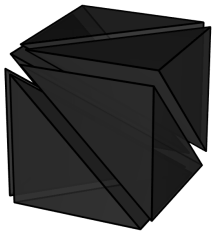
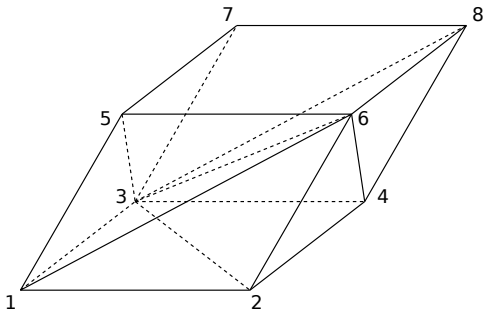


Figure 6: Uniform tetrahedra of equal volume

3.2.3 Six Tetrahedra with uniform shape

In this method, the cube is split into six identical tetrahedra which share a voxel body diagonal as a common edge (Figure 6). Each tetrahedron has two external and two internal faces. The orientation of each of these tetrahedra is different. The most interesting feature here is that the adjoining tetrahedra belonging to adjacent grid cells share a common face. This makes the approach specially suited for the marching tetrahedra.



Index	Vertices
1	1-2-3-6
2	4-2-3-6
3	4-8-3-6
4	7-8-3-6
5	7-5-3-6
6	1-5-3-6

Figure 7: Splitting a general voxel into tetrahedra

Of course, this approach is not only restricted to cubic voxels, but can be applied to any general voxel on a uniform mesh (Figure 7). In case of a general parallelepiped voxel, we have to carry out the split about the shortest body diagonal. This will avoid excessively elongated tetrahedra. The following list enumerates the 6 tetrahedra according to their vertices. Notice that all the tetrahedra share the body diagonal as a common edge and consecutive tetrahedra have a common face.

3.3 Locating a point within a tetrahedron

In the case of a tetrahedron, we define a three dimensional basis, relative to one of the four vertices of the tetrahedron (Figure 9). Let $\boldsymbol{r_i}$ denote the position vector of the vertex i. Let $\boldsymbol{r}$ denote some point inside the tetrahedron. Now we can define three dimensional basis vectors as follows

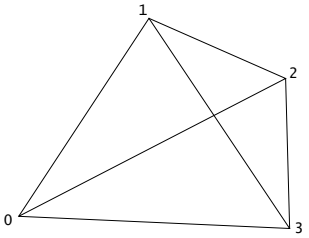


Figure 8: Locating a point inside a tetrahedron

$$\boldsymbol{e_1} = \boldsymbol{r_1} - \boldsymbol{r_0}, \quad \boldsymbol{e_2} = \boldsymbol{r_2} - \boldsymbol{r_0}, \quad \boldsymbol{e_3} = \boldsymbol{r_3} - \boldsymbol{r_0} \tag{3}$$

Then we can write $\boldsymbol{r}$ as

$$\begin{aligned} \boldsymbol{r} &= \boldsymbol{r_0} + \alpha \boldsymbol{e_1} + \beta \boldsymbol{e_2} + \gamma \boldsymbol{e_3} \\ &= (1 - \alpha - \beta - \gamma)\, \boldsymbol{r_0} + \alpha \boldsymbol{r_1} + \beta \boldsymbol{r_2} + \gamma \boldsymbol{r_3} \\ &= (1 - \alpha - \beta - \gamma, \alpha, \beta, \gamma) \end{aligned} \tag{4}$$

The coefficients $(1 - \alpha - \beta - \gamma, \alpha, \beta, \gamma)$ are called the Barycentric co-ordinates of the point $\boldsymbol{r}$. It follows from Equation 4 that if $(\lambda_0, \lambda_1, \lambda_2, \lambda_3)$ are the barycentric co-ordinates of a point $\boldsymbol{r}$, then $\lambda_0 + \lambda_1 + \lambda_2 + \lambda_3 = 1$. The barycentric co-ordinates of a general point $\boldsymbol{r}$ are then calculated by inverting the equation 5.

$$\begin{pmatrix} \boldsymbol{e_1} & \boldsymbol{e_2} & \boldsymbol{e_2} \end{pmatrix} \cdot \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} \boldsymbol{r} - \boldsymbol{r_0} \end{pmatrix} \tag{5}$$

Now, we need to address the uniqueness of this co-ordinate system. This can be done by showing that the above expansion (Equation 4) is obtained starting from any basis formed by the vertices of the tetrahedron. We add and subtract $(1 - \alpha - \beta - \gamma)\, \boldsymbol{r_1}$ from both sides and rearranging, we get

$$\boldsymbol{r} = (1 - \alpha - \beta - \gamma)\, (\boldsymbol{r_0} - \boldsymbol{r_1}) + \beta\, (\boldsymbol{r_2} - \boldsymbol{r_1}) + \gamma\, (\boldsymbol{r_3} - \boldsymbol{r_1}) + \boldsymbol{r_1} \tag{6}$$

which is precisely the representation of $\boldsymbol{r}$ in the basis about $\boldsymbol{r_1}$. Thus the Barycentric Co-ordinate representation is unique.

At this point, it will be useful to note the relationship between the barycentric co-ordinates and the volume of the tetrahedron. For some point P$(\lambda_0, \lambda_1, \lambda_2, \lambda_3)$, inside tetrahedron 0123, the barycentric co-

ordinate associated with vertex i i.e. λ_i is the ratio of the volume of the tetrahedron formed by P and the other three vertices to the volume of tetrahedron 0123. Specifically,

$$\lambda_0 = \frac{V_{P123}}{V_{0123}};\ \lambda_1 = \frac{V_{0P23}}{V_{0123}};\ \lambda_2 = \frac{V_{01P3}}{V_{0123}};\ \lambda_3 = \frac{V_{012P}}{V_{0123}} \tag{7}$$

3.4 Calculating the volume of a tetrahedron

Consider the tetrahedron shown in Figure 9. The scalar triple product of three vectors gives the volume of the parallelepiped defined by them. Our division strategy directs for the creation of tetrahedra of equal volume from each voxel. Then, the volume is simply a fraction of the scalar triple product of the basis vectors

$$V = \frac{1}{6}\left|\boldsymbol{e_1}\cdot(\boldsymbol{e_2}\times\boldsymbol{e_3})\right| = \frac{1}{6}\begin{vmatrix} x_1-x_0 & x_2-x_0 & x_3-x_0 \\ y_1-y_0 & y_2-y_0 & y_3-y_0 \\ z_1-z_0 & z_2-z_0 & z_3-z_0 \end{vmatrix} \tag{8}$$

3.5 Linear Interpolation inside a tetrahedron

The barycentric co-ordinates can be used to define a interpolation inside the tetrahedron

$$f(\boldsymbol{r}) = \lambda_0 f(\boldsymbol{r_0}) + \lambda_1 f(\boldsymbol{r_1}) + \lambda_2 f(\boldsymbol{r_2}) + \lambda_3 f(\boldsymbol{r_3}) \tag{9}$$

Since $\lambda_3 = 1-\lambda_1-\lambda_2-\lambda_3$, $f(\boldsymbol{r}) = const$ is a linear equation in $\{\lambda_0, \lambda_1, \lambda_2\}$ and hence represents a plane in barycentric space. From equation 5 we can deduce that the barycentric space is obtained from a linear transform of real space. Thus we can conclude that given an iso-value $f(\boldsymbol{r})$, the *iso-surface associated with it is planar*.

3.5.1 Uniqueness of the linear interpolation

We have already shown that the barycentric co-ordinates of a point inside a tetrahedron are unique with respect to the vertices of that tetrahedron in Section 3.3. This directly implies that the linear interpolation described in equation 9 is unique with respect to the vertices of the tetrahedron. But we cannot comment on the uniqueness of this interpolation with respect to the choice of tetrahedron in which the interpolation is calculated.

Consider a tetrahedron ABCD as shown in Figure 9. Consider a general point P($\boldsymbol{r}$) inside ABCD. Now we move point A to inside the tetrahedron to point A$'$ such that $\boldsymbol{r}$ is still inside A$'$BCD. Using barycentric co-ordinates, we obtain an expression for the scalar interpolation $f(\boldsymbol{r})$ in terms of A$'$BCD

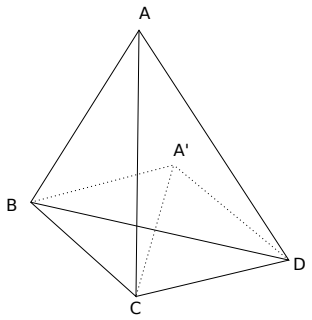


Figure 9: Uniqueness of linear interpolation

$$f(\boldsymbol{r}) = \lambda_{A'} f(\boldsymbol{r_{A'}}) + \lambda_B f(\boldsymbol{r_B}) + \lambda_C f(\boldsymbol{r_C}) + \lambda_D f(\boldsymbol{r_D}) \tag{10}$$

Now, since A$'$ is inside ABCD, we can write an interpolation for $f(\boldsymbol{r_{A'}})$ with respect to ABCD

$$f(\boldsymbol{r_{A'}}) = \eta_A f(\boldsymbol{r_A}) + \eta_B f(\boldsymbol{r_B}) + \eta_C f(\boldsymbol{r_C}) + \eta_D f(\boldsymbol{r_D}) \tag{11}$$

Substituting Equation 11 in Equation 10, we get

$$f(\boldsymbol{r}) = \lambda_{A'}\eta_A f(\boldsymbol{r_A}) + (\lambda_B + \lambda_{A'}\eta_B) f(\boldsymbol{r_B}) + (\lambda_C + \lambda_{A'}\eta_C) f(\boldsymbol{r_C}) + (\lambda_D + \lambda_{A'}\eta_D) f(\boldsymbol{r_D}) \tag{12}$$

Using the areal interpretation of the barycentric co-ordinates (Equation 7), we can write

$$\lambda_{A'} = \frac{V_{PBCD}}{V_{A'BCD}}; \quad \eta_A = \frac{V_{A'BCD}}{V_{ABCD}} \tag{13}$$

Substituting in Equation 12, we get the coefficient of $f(\boldsymbol{r_A})$

$$\lambda_{A'}\eta_A = \frac{V_{PBCD}}{V_{ABCD}} = \Lambda_A \tag{14}$$

which is the barycentric co-ordinate for P with respect to vertex A in tetrahedron ABCD.

Now we can imagine a scenario in which we pull in the vertices A, B, C, D to the vertices A$'$, B$'$, C$'$, D$'$ respectively, one by one. During each pull, we can see that we maintain the uniqueness of the interpolation owing to the analysis above. We end up with the interpolation for $f(\boldsymbol{r})$ in terms of $\{\Lambda_A, \Lambda_B, \Lambda_C, \Lambda_D\}$. This means that the linear interpolation scheme that we have defined is unique,

i.e. the scalar function has the same value no matter with respect to which tetrahedron it is calculated.

This result now gives us the liberty to calculate the volume integral of the interpolated function in a piecewise fashion. Thus, if $V = V_1 + V_2$ then we can write

$$\int_V f(\boldsymbol{r})dV = \int_{V_1} f(\boldsymbol{r})dV + \int_{V_2} f(\boldsymbol{r})dV \tag{15}$$

This result will be used in the following sections.

3.6 Integration over a tetrahedral volume

Once we have established the theory behind the barycentric co-ordinates, it is easy to find the integral of the scalar function over the tetrahedral volume. We need to calculate $\int f(\lambda_0, \lambda_1, \lambda_2)d\lambda_2 d\lambda_1 d\lambda_0$ by putting the suitable limits on each integral. Now each of the λ's varies from 0 to 1, which means that they cover only a scaled volume of the tetrahedron. Thus, we need to explicitly multiply the integral by the tetrahedral volume to obtain the total integral. Even so the values of

$$\begin{aligned} I &= V_{tet} \int_0^1 \int_0^{1-\lambda_0} \int_0^{1-\lambda_0-\lambda_1} f(\lambda_0, \lambda_1, \lambda_2)d\lambda_2 d\lambda_1 d\lambda_0 \\ &= V_{tet} \int_0^1 \int_0^{1-\lambda_0} \int_0^{1-\lambda_0-\lambda_1} \left(\lambda_0 f\left(\boldsymbol{r_0}\right) + \lambda_1 f\left(\boldsymbol{r_1}\right) + \lambda_2 f\left(\boldsymbol{r_2}\right) + \lambda_3 f\left(\boldsymbol{r_3}\right)\right) d\lambda_2 d\lambda_1 d\lambda_0 \end{aligned} \tag{16}$$

always keeping in mind that $\lambda_3 = 1 - \lambda_0 - \lambda_1 - \lambda_2$. Now this integral is analytically soluble and simplifies to

$$I = \frac{V_{tet}}{4} \sum_{i=0}^{3} f\left(\boldsymbol{r_i}\right) \tag{17}$$

where i denotes the vertices of the tetrahedron.

4 Types of iso-surface patches and integrals

The Marching Tetrahedra is a modification of the Marching Cubes algorithm for finding the iso-surface. Each voxel is divided into tetrahedra using the method described above. Then, we look for sign changes in the scalar values along the edges of the tetrahedra thus formed. Let f_{iso} be the iso-value and the function at a vertex i be denoted by $f(\boldsymbol{r_i}) = f_i$. For a particular tetrahedron, the following cases will exist:

1. In the most trivial cases, the value of the function at all four vertices will be either greater than or less than the iso-value (Figure 10). Thus, the iso-surface does not pass through the tetrahedron. Then we have to only determine whether the tetrahedron lies below the iso-surface or not. If the vertices of the tetrahedron all lie below the iso-value, then, we want to integrate over this tetrahedral volume. This accounts for two of the sixteen possible cases. The integral is given by

$$I = \begin{cases} V_{0123} \cdot \left(\frac{f_0+f_1+f_2+f_3}{4}\right); & f_0, f_1, f_2, f_3 \leq f_{iso} \\ 0; & f_0, f_1, f_2, f_3 > f_{iso} \end{cases}$$

2. If just one vertex lies below the iso-value, then we get a scenario as shown in Figure 11. The iso-surface patch is a triangle. We can choose the right part of the split tetrahedron simply by finding out which vertex is inside the iso-surface. This case is equivalent to the case where *three* vertices lie below the iso-value. The integral here is calculated by the difference in the integrals over the bigger and smaller tetrahedra using 15. This accounts for a total of eight cases. The integral here is given by

$$I = \begin{cases} V_{0ABC} \cdot \left(\frac{f_0+3*f_{iso}}{4}\right); & f_0 \leq f_{iso} \\ V_{ABCD} \cdot \left(\frac{f_0+f_1+f_2+f_3}{4}\right) - V_{0ABC} \cdot \left(\frac{f_0+3*f_{iso}}{4}\right); & f_0 > f_{iso} \end{cases}$$

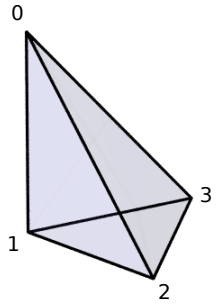


Figure 10: No iso-surface patch

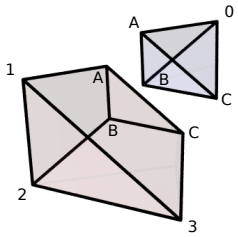


Figure 11: One cutting iso-surface patch

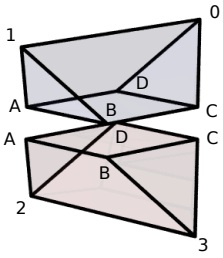


Figure 12: Two cutting iso-surface patches

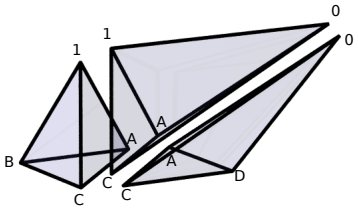


Figure 13: Dividing hexahedron into tetrahedra

3. In the third case, there will be two vertices whose value is below the iso-value. In this case we get two irregular hexahedra (Figure 12). Again, the volume defined by the vertices lower than the iso-value and the iso-surface will be considered in the volume integral. A potential problem here is that although we can still use the sum as shown in equation, we have two irregular hexahedra whose analytical volume integral we don't know. This is solved by dividing the relevant hexahedron into tetrahedra as shown in Figure 13. There are six distinct ways of doing this, although all of them are equivalent mathematically. Now we use the sum of the integrals over each small tetrahedron, as all the vertex scalar values are known. This takes care of the remaining six cases. The integrals are given as

$$I = \begin{cases} V_{1ABC} \cdot \left(\frac{f_0+3*f_{iso}}{4}\right) + V_{01AC} \cdot \left(\frac{f_0+f_1+2*f_{iso}}{4}\right) + V_{0ACD} \cdot \left(\frac{f_0+3*f_{iso}}{4}\right); & f_0, f_1 \leq f_{iso} \\ V_{2ADC} \cdot \left(\frac{f_2+3*f_{iso}}{4}\right) + V_{23AC} \cdot \left(\frac{f_2+f_3+2*f_{iso}}{4}\right) + V_{3ABC} \cdot \left(\frac{f_3+3*f_{iso}}{4}\right); & f_0, f_1 > f_{iso} \end{cases}$$

Thus, the sixteen different possibilities have now boiled down to three general cases. Using the rules enumerated above, it is very easy to find the iso-surface and calculate the volume integrals.

5 Hybrid Marching Cubes and Tetrahedra

We have just discussed two methods of finding the iso-surface using discrete data. We have discussed the marching tetrahedra method as an improvement over the marching cubes method for the reasons mentioned before. Of course, for finding the iso-surface, the marching tetrahedra method is definitely easier to implement. But we can further simplify the calculation of volume integrals by revisiting the marching cubes scheme. Suppose a *voxel* is completely under the iso-surface i.e. all its vertices are below the iso-value. So the constituent tetrahedra are all going to be below the iso-surface. This means that we will need six calculations to find the total integral. A method do directly calculate the integral over the voxel will bring down the number of calculations to just one, provided the order of error in the tetrahedron integral and the voxel integral are comparable. We first start by choosing an interpolation method that is unique over the voxel. We see that the trilinear interpolation in equation 2 is a unique interpolation inside the voxel. Finding the integral over this volume is a simple matter.

$$I_{voxel} = \int_0^1 \int_0^1 \int_0^1 f(x,y,z)dzdydx = \frac{V_{voxel}}{8} \sum_{i=1}^{8} f(\boldsymbol{r_i}) \tag{18}$$

From our voxel $\rightarrow$ tetrahedra division strategy, we know that $V_{voxel} = 6V_{tet}$. So

$$I_{voxel} = \frac{3V_{tet}}{4} \left(f_1 + f_2 + \cdots + f_8\right) \tag{19}$$

Now if we calculate the integral by summing the integrals over the six constituent tetrahedra, we see that the scalars at all six vertices do not have equal weight, with the vertices of the body diagonal having different weights from the rest.

$$I_{tet} = \frac{V_{tet}}{2}(f_1 + f_2 + f_4 + f_5 + f_7 + f_8) + \frac{3V_{tet}}{2}(f_3 + f_6) \tag{20}$$

In both the equations, we see that the sum of the coefficients of each vertex scalar is $6V_{tet}$, meaning that the two equations are comparable and compatible with each other, thus making the hybrid scheme possible.

6 Notes about the code

The code was written in C++. The method used was the hybrid cubes + tetrahedra method. The code uses GAUSSIAN CUBE format files for reading in the mesh and the scalar data.

The algorithm has two distinct levels. First it runs through all the voxels and does two things

1. When all the vertices of a voxel are below the iso-surface, it calculates the integral over these voxels and adds it to the the total integral.
2. If there is a sign change along any one edge of the voxel, then this voxel contains the required iso-surface. We split this voxel into tetrahedra and store the data for each. Then the tetrahedron routine calculates al the iso-surface patches and their contributions to the integral.
3. If all the vertices of the voxel are above the iso-value then the voxel is ignored.

The direct mode takes in the iso-value from the user and plots the iso-surface and calculates the volume integral. In the inverse mode, the user gives the required volume integral and the program finds the corresponding iso-value and plots the surface that encloses this integral. Once all the calculations are done, the code also implements a small routine using OpenGL that displays an interactive window (screenshot in Figure 14), where the iso-surface can be viewed and modified.

7 Sample Results

The first part of the task is to visualise the iso-surfaces for given input functions. As a test case, we choose a simple hydrogenic wave function defined by $f(r) = 2e^{-r}$. For this function we choose various grid sizes for a simple cubic mesh and plot the iso-surfaces for some chosen iso value. The results can be seen in Figure 15. The approximated iso-surface is plotted with the actual analytically calculated iso-surface (in this case a sphere). It is clear that the accuracy of the result decreases drastically with grid size as can be seen from the figure.

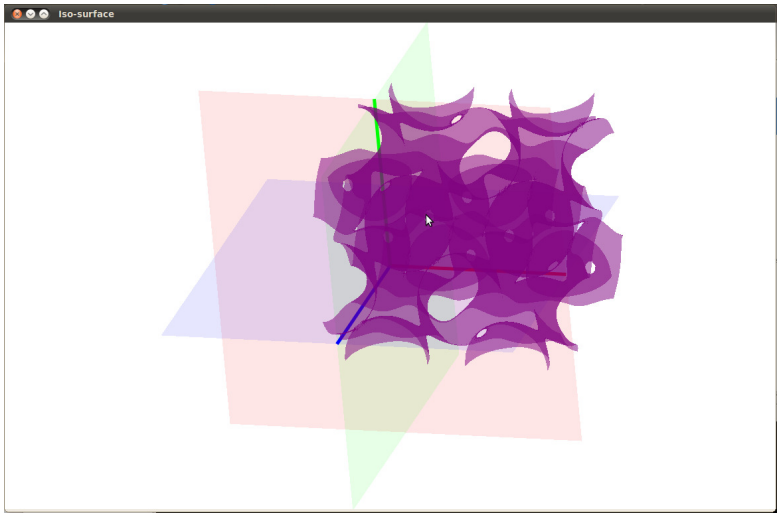


Figure 14: Iso-surface

8 Outlook

In summary, we now have a technique to visualise a certain class of discrete data and additionally calculate relevant properties from this data. Of course, there are many further ideas that immediately come to mind. In this work, we mainly relied on uniform meshes which then simplified the marching tetrahedron algorithm. However, there may be a possibility to also use unstructured data with this algorithm. Secondly, we see that the algorithm is more or less repetitive, i.e. the same process is executed on large chunks of data. This immediately makes it a candidate for parallelisation, so that one can handle even finer meshes and also shorten the execution times required for the program. The visualisation is already implemented in OpenGL. Since graphics is such an important part of the program, there is also the possibility to use general purpose graphics processing units to handle the parallelisation. All these point remain to be investigated and form the basis for future work in this context.

9 Acknowledgements

At the outset, I would like to thank the Forschungszentrum Jülich and in particular, Jülich Supercomputing Center for giving me the opportunity to participate in this wonderful program. It has definitely widened my understanding of science in general, and has given me a unique insight into the world of simulation and supercomputing. I would like to thank my advisor Prof. Dr. Erik Koch for guiding me throughout and helping me complete my task to the best of my abilities. I would like to thank Mathias Winkel and Natalie Schröder for the impeccable organisation and all the other fun things we did during these two months. Last, but by no means the least, I would like to thank my fellow guest students. The program would not have been as enriching without their presence.

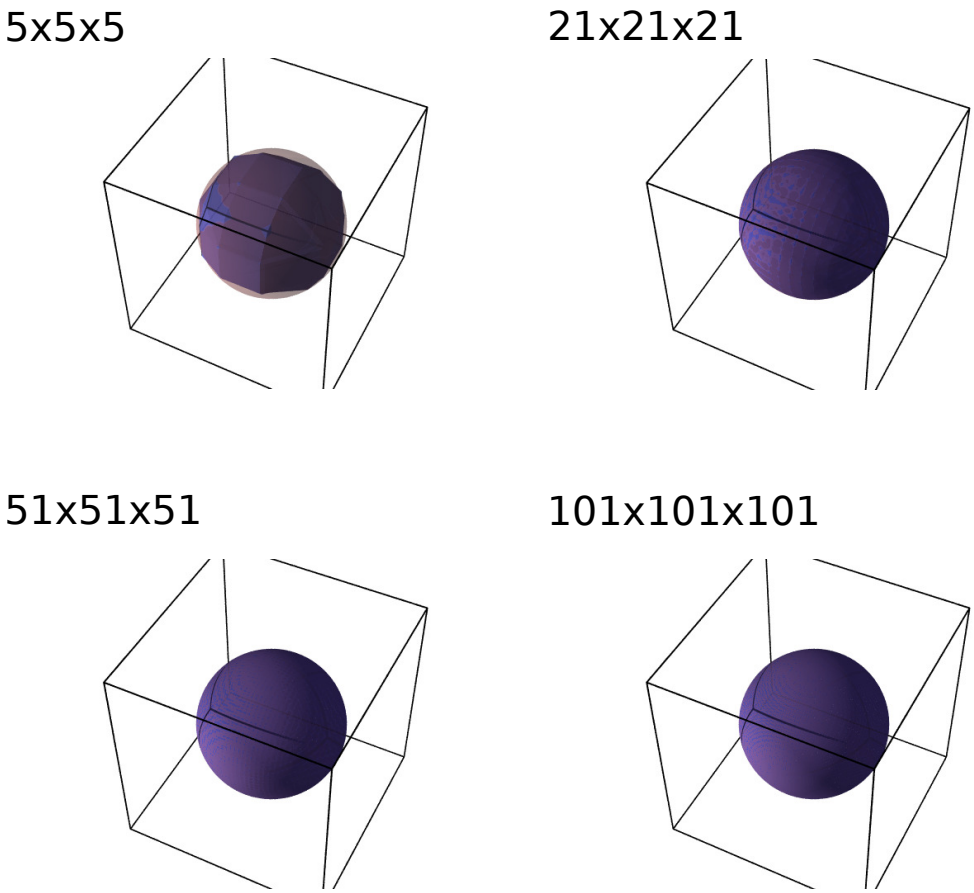


Figure 15: Iso-surface vs. Grid size

References

1. "Polygonising a Scalar Field": http://www.paulbourke.net/geometry/polygonise/
2. Improved tetrahedron method for Brillouin-zone integrations, P.E. Blöchl, O. Jepsen, O.K. Andersen, Phys. Rev. B 49 (1994) 16223.
3. POV-Ray - Persistence of Vision Raytracer: http://www.povray.org/
4. OpenGL; http://www.opengl.org/
5. GLUT - The OpenGL Utility Toolkit: http://www.opengl.org/resources/libraries/glut/
6. GLT ZPR - Library for Pan, Rotate and Zoom: http://www.nigels.com/glt/gltzpr/
7. Wolfram Mathematica: http://www.wolfram.com/

Quantum Chemical Calculations on the Potential Energy Surface of Ozone

Janine George

RWTH Aachen University
Faculty of Mathematics,
Computer Science and Natural Sciences
Templergraben 55
52062 Aachen

E-mail: janine.george@rwth-aachen.de

Abstract:
This work focuses on the quantum chemical calculation of the ground state energy surface of ozone and, especially, on the minimum energy path within the dissociation threshold for the reaction $O_3(^1A_1) \longrightarrow O(^3P) + O_2(^3\Sigma_g^-)$. In order to improve these quantum chemical calculations, mainly internally contracted Multi-reference Averaged Quadratic Coupled-Cluster (ic-MR-AQCC) and internally contracted Multi-reference Configuration Interaction with all Single and Double excitations with Davidson or Pople correction (ic-MR-CISD+Q_D/+Q_P) energies were compared with MR-AQCC and MR-CISD+Q_D/+Q_P energies. The barrier of the minimum energy path cut disappears by the application of the uncontracted methods instead of the internally contracted methods. This is explainable with the consideration of a higher amount of electron correlation energy. Uncontracted MR-AQCC and MR-CISD+Q_D/+Q_P overestimate the experimental dissociation energy of 1.143 eV even at the finite cc-pV5Z basis set by 0.030 eV. While the basis set superposition error (BSSE) can be ruled out as a source for this discrepancy, size-consistency corrections may be considered as a possible error source.

1 Introduction

Ozone was proposed in 1840 by Schönbein. [1] Ozone has been identified as an important molecule in atmospheric physics and in climate applications. Several properties and reactions of ozone are still unclarified, for instance the isotope enrichment effect. In the upper atmosphere, an equal enrichment of ^{17}O and ^{18}O over ^{16}O within the formation of ozone from the recombination of oxygen atoms and molecules was found in contrast to the standard, mass-dependent value. [2] In order to get a better understanding of ozone, its properties and, primarily, this isotope enrichment effect, theoretical calculations of the kinetics of the dissociation $O_3(^1A_1) \longrightarrow O(^3P) + O_2(^3\Sigma_g^-)$ and, therefore, the potential energy surface (PES) are highly relevant.
The PES so far calculated by ab initio methods leads to a theoretical isotopic exchange rate coefficient

which is too small compared with the experimental value. [3] The question arises whether there is a barrier on the dissociation minimum energy path, since this barrier influences the reaction dynamics and leads to a smaller rate coefficient of the dissociation. All ab initio calculations on the minimum energy path show an activation barrier at the entrance of the dissociation channel at the O-O bond lengths $r_1 \approx 2.28$ a.u. and $r_2 \approx 3.8$ a.u. and the O-O-O angle $\alpha \approx 115 - 117°$ until now. Furthermore, these calculations predict a van der Waals minimum along the dissociation reaction coordinate at $r_2 = 4.5 - 5.0$ a.u. and the previous given values of r_1 and α. [3] Besides, also an analytical PES exists, which was constructed from high-resolution infrared spectra. However, this PES must be validated and improved by ab initio calculations, since experimental data is not yet available, especially, close to the transition states. [4, 5, 3]
Holka et al. [3] recalculated the minimum energy path cut with ic-MR-AQCC, ic-MR-CISD+Q_D/+Q_P and showed that the core electron effect is negligible. Moreover, Holka et al. [3] assumed the approximation within the internally contracted methods could lead to the existing barrier and van der Waals minimum. Thus, a main part of this work is comparing internally contracted MR-AQCC-method and uncontracted MR-AQCC-method along the dissociation minimum energy path.

2 Theory

2.1 General Introduction to Quantum Chemistry

The theoretical calculations are based on the solution of the non-relativistic, time independent Schrödinger equation: [6, 7]

$$\hat{H}\Psi = E\Psi. \tag{1}$$

The Hamiltonian $\hat{H}$ corresponds to the total energy of a system and is an analogue to the classical Hamiltonian. The wave function Ψ is the description of the quantum state of a molecule. E is the total energy of the system. The Hamiltonian includes the kinetic and potential energy operators:

$$\hat{H} = \sum_i \frac{1}{2}\cdot\nabla_i^2 + \sum_A \frac{1}{2}\cdot\nabla_A^2 - \sum_{iA}\frac{Z_A}{r_{iA}} + \sum_{i\neq j}\frac{1}{r_i - r_j} + \sum_{A\neq B}\frac{Z_A Z_B}{r_A - r_B}. \tag{2}$$

$\sum_i \frac{1}{2}\cdot\nabla_i^2$ describes the electronic kinetic energy and $\sum_A \frac{1}{2}\cdot\nabla_A^2$ the nuclear kinetic energy. $\sum_{i\neq j}\frac{1}{r_i - r_j}$ is the electronic, $\sum_{A\neq B}\frac{Z_A Z_B}{r_A - r_B}$ the nuclear Coulomb-interaction and $\sum_{iA}\frac{Z_A}{r_{iA}}$ the Coulomb-interaction of electrons and nuclei.
In the Born-Oppenheimer approximation [7], the total wave function $\Psi(\boldsymbol{R})$ is written as a product of the electronic wave function $\Psi(\boldsymbol{r})$ and nuclear wave function $\Psi(\boldsymbol{R})$:

$$\Psi(\boldsymbol{R}, \boldsymbol{r}) = \Psi(\boldsymbol{r})\cdot\Psi(\boldsymbol{R}). \tag{3}$$

The coupling of electrons and nuclei is neglected, since −simplified− the electrons can react really fast on the movement of the nucleus due to the high difference in mass. The eigen value of the resulting electronic Schrödinger-equation E_{el} as a function of the nuclear coordinates describes the potential energy surface (PES) [7]. A point of the PES corresponds to a molecular configuration during a reaction. In practice, the PES is calculated pointwise and it is often adequate to calculate only minima and saddle

points (= transition states). The Schrödinger equation is only exactly solvable for molecules like H_2^+. For systems with more electrons or nuclei, numeric approximations are needed. Relativistic effects are relevant for core-electrons which are moving at a substantial fraction of the speed of light.
The Pauli-principle [8, 7] demands the antisymmetry of the wave function. The antisymmetry principle demands that a wave function changes its sign upon permutation of the coordinates of any pair of electrons. Consequently, the total n-electron wave function $\Psi(\tau_1, \tau_2, ..., \tau_N)$ is expanded as an antisymmetrized product of one electron functions ($\chi(\tau)$) (Slater-determinants):

$$\Psi(\tau_1, \tau_2, ..., \tau_N) = \frac{1}{\sqrt{N!}} \begin{vmatrix} \chi_1(\tau_1) & \chi_2(\tau_1) & ... & \chi_N(\tau_1) \\ \chi_1(\tau_2) & \chi_2(\tau_2) & ... & \chi_N(\tau_2) \\ ... & ... & ... & ... \\ \chi_1(\tau_N) & \chi_2(\tau_N) & ... & \chi_N(\tau_N) \end{vmatrix}. \tag{4}$$

These one electron functions $\chi(\tau)$ consist of a spin function $\sigma(\omega)$ (which represents either spin-up or spin-down states) and a spatial single electron function $\psi(r)$, whereas τ describes position x and spin variables ω of a single electron:

$$\chi(\tau) = \psi(x)\sigma(\omega). \tag{5}$$

2.2 Variational Principle

The methods described in this report are all variational [7]. The expectation $E = \frac{<\Psi|\hat{H}|\Psi>}{<\Psi|\Psi>}$ which is calculated with a trial wave function Ψ is always greater than or equal to the exact total energy E_0. The lower the expectation value of the energy, the more accurate is the (ground-state) wave function.

2.3 Hartree-Fock-Approximation

In the Hartree-Fock-Approximation [7] the wave function is described with just a single Slater determinant. Using the LCAO-MO-ansatz (see Eq. 7) with Gaussian functions Φ_μ and LCAO-MO-coefficients $c_{\mu i}$ for the description of the spatial orbitals and minimizing the total energy with respect to all variational parameters (see Eq. 8) under orthonormalization constraints (cf. Eq. 9) the Roothaan-Hall equations [9, 10] are received:

$$FC = SC\epsilon. \tag{6}$$

F is the Fock matrix, C the matrix of the LCAO-MO-coefficients, S the overlap matrix of the basis functions and ϵ the diagonal matrix of MO energies.

$$\psi_i = \sum_{\mu}^{K} c_{\mu i}\Phi_\mu \tag{7}$$

$$\frac{\partial E}{\partial c_{\mu i}} = 0 \tag{8}$$

$$< \Psi_i | \Psi_j >= \delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{9}$$

A basis set is determined by the choice of Gaussian functions. The number of the applied Gaussian functions K (see Eq. 7) increases the computing time and the accuracy. The limit which can be reached in the Hartree Fock approximation with an infinite basis set is called Hartree Fock limit. The difference of the exact non-relativistic energy and this Hartree Fock limit defines the electron correlation energy. Going beyond the HF approximation, we need to account for the contributions of the other possible configurations.
The electron correlation [11, 12] is commonly separated into its static and dynamic component. The static electron correlation considers near-degeneracy effects while dynamic correlation accounts for the remainder. Thus, to account for static electron correlation is important for the description of bond breaking.

2.4 Configuration Interaction

The Hartree Fock approximation can be improved by choosing a more flexible wave function as in the CI [7] (= Configuration Interaction) method. The CI-method expands wave function in terms of determinants grouped by their excitation level Ψ_i,Ψ_{ij}, etc. (single, double, etc. excitations) with respect to the ground state (Ψ_0). In the case of the Single Reference-CI this ground state is described with the Hartree Fock determinant. The wave function is constructed as follows:

$$\Psi = \tilde{c}_0 \Psi_0 + \sum_{ia} {}' \tilde{c}_{ia} \Psi_i^a + \sum_{ijab} {}' \tilde{c}_{ijab} \Psi_{ij}^{ab} ... + ... \, . \tag{10}$$

The CI-coefficients $\tilde{c}_i$ are determined by variationally minimizing the energy eigenvalue.
If the sum is not truncated all possible determinants are included and this method is denoted Full Configuration Interaction (FCI). FCI is the exact result with a given one electron basis set and the other methods are an approximation of the FCI limit. Due to the high computational cost, Equation 10 is usually truncated to just single (S) or single and double (SD) excitations. The main problem of these truncated CI methods is that they are not size consistent and extensive. Size consistency [11] requires that the total energy of two non-interacting molecules/atoms A and B does not depend on whether it is computed independently E_A and E_B or jointly as supermolecule E_{A+B}. Size extensivity [11] is a more general mathematical concept implying the correct energy scaling with the number of electrons.

2.4.1 Multi-configuration Self-consistent Field and Multi-reference Configuration Interaction

In the Multi-configuration Self-consistent Field (MCSCF) [11] method not only the CI coefficients but also MO-coefficients are variationally optimized. The optimization procedure is iterative as in the Hartree Fock method. The adequate selection of the configurations is the major problem of the MCSCF methods. The Complete Active Space Self-consistent Field (CASSCF) partitions the MOs into virtual (unoccupied), active (arbitrary occupation) and inactive (doubly occupied) orbital subspaces and is the

Table 1: Test of the size-consistency of the internal contracted, used methods

basis set	ic-MR-AQCC (cm^{-1})	ic-MR-CISD (cm^{-1})	ic-MR-CISD+Q_D (cm^{-1})	ic-MR-CISD+Q_P (cm^{-1})
cc-pVTZ	91.52	4514.54	926.72	-217.77
cc-pVQZ	117.17	5031.21	1147.46	-158.77
cc-pV5Z	124.48	5184.49	1226.61	-125.54
cc-pV6Z	126.45	5233.05	1252.77	-112.27

most popular approach in this field.
As in MCSCF, the Multi-reference Configuration Interaction Method [13] commonly defines a reference configuration space through orbital occupation restrictions which accounts for near-degeneracy effects. Including all singly and doubly excited configurations (MR-CISD) with respect to the reference configuration space allows for dynamic electron correlation effects. Higher excited configurations are usually omitted. The coefficients of the wave function expansion are optimized variationally. This general method is applicable to all types of problems including bond-breaking.
MR-CISD as all truncated CI expansions are not size consistent and extensive [11]. Several approaches [14] have been developed to approximately reinstate size-extensivity. The approaches can be divided into a posteriori corrections (Davidson−type or Pople-type corrections to MR-CISD (MR-CISD-Q_D/Q_P)) and methods that modify the CI functional (MR-AQCC (Multi-reference averaged quadratic coupled-cluster method)).
An approximation to the conventional MR-CISD methods are the internally contracted (ic) variants thereof. The latter treat the (MCSCF) reference wavefunction as an entity to which the excitations are applied, thereby reducing the number of independently variationally optimized parameters drastically to the same order of magnitude as for SR-CI.
In order to test the size consistency of the ic-MR-CISD-methods and to explain the preference for ic-MR-AQCC and ic-MR-CISD+Q_D and its uncontracted (uc) counterparts uc-MR-AQCC and uc-MR-CISD+Q_P for the calculation of the PES, the size consistency error $E_{Size-consistency}$ was calculated with the following equation:

$$\Delta E_{Size-consistency} = E((O_2 + O)(^1A_1)) - E(O_2(^3\Sigma_g^-)) - E(O(^3P)). \tag{11}$$

$E(O_2 + O)$ is the energy calculated with an infinite distance of the two fragmental monomers, $E(O_2)$ and $E(O)$ are the energies of the monomers with the same geomety and C_S symmetry. The infinite distance was approximated with $r_2 = 15$ a.u., the other values were $r_1 = 2.275$ a.u. and $\alpha = 117°$. The results are displayed in Table 1. The size-consistency-errors of ic-MR-AQCC and ic-MR-CISD+Q_P are $126.45\ \mathrm{cm}^{-1}$ and $-112.27\ \mathrm{cm}^{-1}$ and, therefore, the smallest as expected.

2.5 Basis Sets

In this work, the correlation consistent basis sets [15, 16, 17], developed by Dunning and coworkers, were applied. These basis sets recover the correlation energy of valence electrons. Basis functions, which contribute a similar amount of correlation energy, are introduced simultaneously. That results in the composition displayed in Table 2. Due to the fact that there is a basis set convergence error [18]

in any finite basis set, an extrapolation to the complete basis set limit was applied. The extrapolation scheme by Halkier et al. was used [19]:

$$E_{CORR}(X) = E_{CORR}^{CBS} + \frac{const}{X^3}. \tag{12}$$

X is the cardinal number of the basis as in cc-pVXZ, $E_{CORR}(X)$ is the electron correlation energy in basis set X, E_{CORR}^{CBS} the electron correlation energy at the a complete basis set limit and $const$ a constant. Within multireference methods the definition of the electron correlation energy is somewhat vague. In this work, the correlation energy is rated

$$E_{CORR}(X) \approx E_{MR-AQCC}(X) - E_{MCSCF}(X), \tag{13}$$

although this definition neglects the static electron correlation already contained in the $E_{MCSCF}(X)$. Apart from the basis set convergence error, there is also the basis set superposition error (BSSE) [18].

Table 2: Correlation Consistent Basis Sets

Basis set	Primitive functions	Contracted functions
cc-pVDZ	9s,4p,1d	3s,2p,1d
cc-pVTZ	10s,5p,2d,1f	4s,3p,2d,1f
cc-pVQZ	12s,6p,3d,2f,1g	5s,4p,3d,2f,1g
cc-pV5Z	14s,9p,4d,3f,2g,1h	6s,5p,4d,3f,2g,1h
cc-pV6Z	16s,10p,5d,4f,3g,2h,1i	7s,6p,5d,4f,3g,2h,1i

This error arises when e.g. two fragments A and B approach to form a supermolecule. Therby, the description of fragment B is improved by the use of basis functions of fragment A and vice versa. The basis set of two isolated fragments is, on the contrary, not extended. Thus, this BSSE leads to an error of calculated energy difference, especially if bond breaking is treated as in the calculation of the dissociation energy.

2.6 Technical Details

In order to perform ic-MR-AQCC [20] or ic-MR-CISD [21, 22], MOLPRO [23] was used. For the calculations of ozone, a two-stage MCSCF-calculation [24, 25] was done. At some geometrical configurations also some of the valence orbitals are doubly occupied resulting in an uncontrolled mixing of core and valence orbitals and thus to non continuous PES, so first the 1s orbitals were determined in a smaller MCSCF calculation step with all six nearly doubly occupied orbitals in the inactive space andd 12 electrons in the active space. The second step used a full valence CAS and a frozen core approximation. The core part of the wave function changes only slightly during the dissociation and is not responsible for the outstanding discrepancies between theory and experiment, which follows, for example, from calculations of Holka et al. [3]. For all MR-CI-calculations a full valence CAS as a reference space and also a frozen core approximation was applied. A-posteriori size-extensivity corrections of Davidson-type [26, 27] and Pople-type [28] have been computed. Scalar relativistic corrections for mass-velocity and Darwin-terms have been evaluated from first order perturbation theory. Fully compatible MR-AQCC and MR-CISD calculations have been computed with the COLUMBUS [29, 30, 31, 32] program package.
Molpro-calculations have been carried out in reduced point group symmetry C_s, while Columbus cal-

culations used the highest possible abelian point group symmetry throughout (C_s, C_{2v}). For molecular or atomar oxygen the number of electrons in CAS classes was adapted to the total number of electrons.

3 Results

3.1 ic-MR-AQCC-method

In order to investigate the convergence of the applied ic-MR-AQCC-method with an extended basis set and recieve an accurate and convergent MEP-Cut, several 1D-Cuts within the PES beside the MEP-Cut were calculated (see Table 3 for parameters). These 1D-Cuts were fitted in a least square fit to a polyno-

Table 3: Structural data of ozone for the computed 1 D cuts

1D Cut	r_1 (a.u.) (bond length O-O)	r_2 (a.u.) (second bond length O-O)	α (°) (angle O-O-O)	number of points
Minimum Energy Path Cut (MEP-CUT)	2.275	3.400−15.000	117.0	13
Barrier Bending Cut	2.275	3.800	102.5−130.0	6
Barrier Stretch Cut	2.200−2.400	3.800	117.0	8
Van-der-Waals Minimum Bending Cut	2.275	5.000	102.5−130	6
O_3 Stretch Cut	2.400	2.000−4.000	117.0	12
Dissociation Valence Cut	2.225−2.350	10	117.0	6
Dissociation Angle	2.275	10.000	102.5−130.0	3
C_{2v} Symmetric Cut	2.200−3.000	2.20−3.00	117.0	9
C_{2v}Bending Points	2.400	2.400	95.0−145.0	22

mial of n'th order to judge the smoothness of the computed PES [3]:

$$E(X) = E_0 + F_{XX}(X - X_0)^2 + F_{XXX}(X - X_0)^3 + F_{XXXX}(X - X_0)^4 + \dots . \qquad (14)$$

X represents the O-O bond length or O-O-O bond angle in a.u.; X_0 is the minimum along the chosen coordinate; E_0, F_{XX}, etc. are parameters obtained from the least square fit and F_{xx} etc. are identified with the force constants. $E(X)$ are energies relative to the minimum energy point in the input data.
The dissociation angle path was omitted with the ic-MR-AQCC-method, since only three points of the dissociation angle path were calculated. The fitting results of ic-MR-AQCC-R_{MVD} (relativistically corrected) are given in Table 4. The obtained parameters converge with an extended basis set and hence, the energies of each calculated point converge. The following presented accuracy is the maximal difference of the parameters calculated with cc-pV6Z and cc-pV5Z basis sets. The force constants F_{XX}, F_{XXX}, F_{XXXX} converge to an accuracy of $410\ \mathrm{cm}^{-1}\mathrm{a.u.}^{-2}$ respectively $0.03\ \mathrm{cm}^{-1}\mathrm{deg}^{-2}$, $620\ \mathrm{cm}^{-1}\mathrm{a.u.}^{-3}$ respectively $0.001\ \mathrm{cm}^{-1}\mathrm{deg}^{-3}$ and $720\ \mathrm{cm}^{-1}\mathrm{a.u.}^{-4}$ respectively $0.0001\ \mathrm{cm}^{-1}\mathrm{deg}^{-4}$ and x_0 to an accuracy of 0.002 a.u. respectively $0.1\ °$.
Moreover, the with a cc-pV5Z/cc-pV6Z basis set calculated Minimum Energy Path includes an activation barrier at $r_2 = 3.9$ a.u. and a van der Waals minimum at $r_2 = 5.00$ a.u., as already calculated with augmented cc-pVXZ basis sets by F. Holka et al. [3].
The calculated dissociation energy should be compared with the experimental values and, moreover, the barrier, the energy of the maximum relative to that of the dissociation limit and the energy of the

van der Waals minimum relative to that of the dissociation limit should be compared to the tendencies following from the experimental kinetics of the dissociation.
The dissociation energy was calculated as the energy difference of the MEP cut structure at $r_2 = 15$ a.u. and the equilibrium geometry $r_1 = r_2 = 2.4$ a.u. and $\alpha = 117°$. Table 5 compiles the dissociation energies with different basis sets and the CBS limits computed at ic-MR-AQCC level of theory. Even with the cc-pV6Z basis set the computed dissociation energy (1.099 eV) is 0.044 eV below the experimental value [3] of 1.143 eV. Basis set extrapolation reduces the discrepancy to 0.014 eV while relativistic corrections reduce the dissociation energy slightly by 0.003 eV.
The barrier along the MEP-Cut decreases with increasing basis set size. The barrier height is defined as the energy difference of the energy maximum and the Van der Waals minimum as displayed in Table 6. The barrier even does not vanish at the CBS limit. For basis sets larger than the cc-pVQZ basis the barrier maximum drops below the dissociation limit. This tendency is also shown in the work of F. Holka et al. [3].
Additionally, the van der Waals minimum deepens relative to the dissociation limit with increasing basis set size (see Table 6), since diffuse basis functions are needed to describe a van der Waals minimum better. The relativistic correction has only a slight impact on the dissociation energy (-0.003 eV) and on the barrier (≈ 10 cm^{-1}).

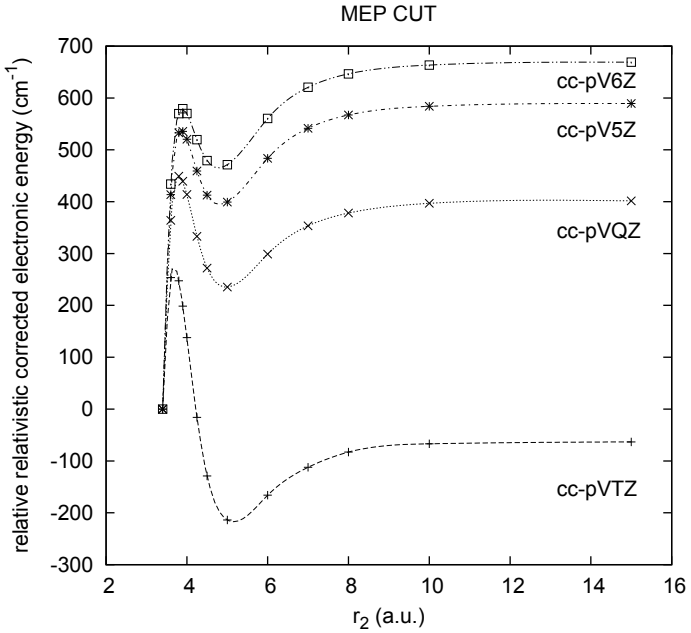


Figure 1: Minimum Energy Path Cut was calculated with the ic-MR-AQCC method and different basis sets. The energy in each MEP Cut is relative to the ozone energy at $r_1 = 2.275$ a.u., $r_2 = 3.4$ a.u. and $\alpha = 117°$.

3.2 ic-MR-AQCC vs. MR-AQCC-method

The internal contraction reduces the number of variational parameters by a factor of ≈ 100 relative to the uncontracted method and ties the CI wave function somewhat to the quality of the MCSCF wave function. It is expected that especially around the barrier this may lead to qualitative and semi-quantitative errors. Thus, the contraction error is assessed by comparing ic-MR-AQCC with uc-MR-

Table 4: Comparison of the parameters of the fitted 1D-Cuts calculated with different basis sets and with the ic-MR-AQCC+R_{MVD}-method

1 D -Cut	E_0 (cm^{-1})	F_{XX}	F_{XXX}	F_{XXXX}	x_0 (a.u.ldeg)	N_{points}	$N_{parameters}$	rms (cm^{-1})
Barrier Bending Cut								
cc-pVTZ	-8.23	2.76912	-0.03494	0.00022	115.248	6	5	0.13
cc-pVQZ	-8.18	2.82027	-0.03415	0.00017	115.272	6	5	0.11
cc-pV5Z	-8.49	2.82490	-0.03384	0.00018	115.240	6	5	0.13
cc-pV6Z	-8.67	2.83339	-0.03403	0.00018	115.225	6	5	0.11
Barrier Stretch Cut								
cc-pVTZ	-8.19	78742	-103633	87343.6	2.28973	8	7	0.00094
cc-pVQZ	-2.66	80480	-106053	88055.9	2.28072	8	7	0.00037
cc-pV5Z	-1.01	80792	-106194	87334.5	2.27853	8	7	0.00070
cc-pV6Z	-0.54	80971	-106503	87990.3	2.27759	8	7	0.00066
Van-der-Waals Minimum Bending Cut								
cc-pVTZ	0.01	0.25618	-0.00339	0.00003	116.895	6	5	0.03
cc-pVQZ	0.01	0.27187	-0.00371	0.00003	117.094	6	5	0.02
cc-pV5Z	-0.02	0.27905	-0.00359	0.00002	117.318	6	5	0.02
cc-pV6Z	-0.04	0.27481	-0.00347	0.00002	117.419	6	5	0.02
O3 Stretch Cut								
cc-pVTZ	-33.10	39597	-60268	54677	2.42878	12	11	2.25
cc-pVQZ	-4.73	41842	-63666	59557	2.41174	12	11	2.18
cc-pV5Z	-1.40	42428	-64334	60598	2.40769	12	11	2.16
cc-pV6Z	-0.37	42664	-64658	61123	2.40591	12	11	2.16
Dissociation Valence Cut								
cc-pVTZ	-1.55	80584	-105732	92472	2.29560	6	5	0.00002
cc-pVQZ	-12.28	82385	-108404	91419	2.28711	6	5	0.00001
cc-pV5Z	-8.62	82709	-108628	90599	2.28514	6	5	0.00003
cc-pV6Z	-7.09	82921	-108943	90599	2.28419	6	5	0.00002
C_{2v} **Symmetric Cut**								
cc-pVTZ	-50.38	102880	-138070	124302	2.42181	9	8	0.00104
cc-pVQZ	-9.41	106331	-144515	129357	2.40935	9	8	0.00397
cc-pV5Z	-4.07	107430	-146058	130528	2.40613	9	8	0.00834
cc-pV6Z	-2.42	107832	-146672	131247	2.40472	9	8	0.00286
C_{2v} **Bending Points**								
cc-pVTZ	-0.39	16.56920	-0.17172	0.00226	116.859	22	8	0.12
cc-pVQZ	1.24	16.35920	-0.17003	0.00313	116.864	22	8	2.42
cc-pV5Z	0.85	16.35620	-0.17004	0.00296	116.858	22	8	2.22
cc-pV6Z	0.94	16.32990	-0.16942	0.00294	116.871	22	8	2.29

Table 5: **a)** Dissociation energy from ic-MR-AQCC and ic-MR-AQCC+R_{MVD} (Experimental value [3] corrected by the value for Zero Point Energy (ZPE) of O_2 and O_3: 1.143 eV) **b)** Difference of the energy maximum at $r_2 = 3.6$ a.u. (cc-pVTZ), $r_2 = 3.8$ a.u. (cc-pVQZ) respectively $r_2 = 3.9$ a.u. (cc-pV5Z/cc-pV6Z) and the dissociation limit energy at $r_2 = 15$ a.u. within the MEP-Cut

basis set	a) Dissociation energy		b) Barrier top relative to dissociation limit	
	ic-MR-AQCC (eV)	ic-MR-AQCC+R_{MVD} (eV)	ic-MR-AQCC (cm^{-1})	ic$-$MR$-$AQCC+R_{MVD} (cm^{-1})
cc-pVTZ	0.887	0.881	321.02	316.91
cc-pVQZ	1.022	1.019	57.43	47.46
cc-pV5Z	1.078	1.075	-46.94	-53.82
cc-pV6Z	1.099	1.096	-83.00	-89.81
(Q,5)	1.129	1.126	-152.23	-157.23
(5,6)	1.126	1.123	-144.68	-151.37

Table 6: **a)** Difference of the van der Waals minimum at $r_2 = 5$ a.u. and the dissociation limit energy at $r_2 = 15$ a.u. within the MEP-Cut **b)** Barrier calculated with the MEP-Cut especially with the maximum at $r_2 = 3.6$ a.u. (cc-pVTZ), $r_2 = 3.8$ a.u. (cc-pVQZ) respectively $r_2 = 3.9$ a.u. (cc-pV5Z/cc-pV6Z) and the van der Waals minimum at $r_2 = 5$ a.u.

basis set	a) van der Waals minimum relative to dissociation limit		b) Barrier height	
	ic-MR-AQCC (cm^{-1})	ic$-$MR$-$AQCC+R_{MVD} (cm^{-1})	ic-MR-AQCC (cm^{-1})	ic$-$MR$-$AQCC+R_{MVD} (cm^{-1})
cc-pVTZ	-151.32	-150.55	472.34	467.46
cc-pVQZ	-165.00	-166.27	222.43	213.73
cc-pV5Z	-189.41	-190.07	142.46	136.25
cc-pV6Z	-197.13	-197.84	65.75	60.77
(Q,5)	-217.97	-218.00	65.75	60.77
(5,6)	-217.48	-218.26	72.80	66.88

AQCC using the cc-pVTZ basis set. As in Section 3.1 the 1D-Cuts were fitted with a polynomial expansion and the resulting force constants are compared (see Tab. 7). The force constants at uc-MR-AQCC are systematically lower by 8% than their ic-MR-AQCC counterparts. The exception is the van der Waals minimum bending cut ($< 15\%$), since these force constants are small in comparison to the others. The difference in α_0 and r_0 is always smaller than 0.4 % (0.009 a.u., 0.214 °). The largest difference is in the parameter E_0, as expected, since this parameter is the statistically most uncertain fit parameter. As an example for the fitted 1D-Cuts, the C_{2v} symmetric cut is displayed in Fig. 2.
The MEP-Cut reveals qualitatively different predictions for the two different methods (Fig. 2). The barrier has almost vanished at the MR-AQCC/cc-pVTZ level (barrier height 65.79 cm^{-1}) and, in addition, the barrier top has already dropped below the dissociation limit (-156.31 cm^{-1}). The position of the vdW minimum relative to the dissociation limit has decreased by 71 cm^{-1} to -222.10 cm^{-1} relative to the ic-MR-AQCC values (see Tables 5 and 6). This is somewhat surprising as the vdW minimum should be equally described with both variants. Since the barrier height and the dissociation energy is far from converged with a cc-pVTZ basis, the MEP-Cut had to be recalculated with uc-MR-AQCC-method and larger basis sets (cc-pVQZ, cc-pV5Z).
A graphical comparison of uc-MR-AQCC-method and its ic counterpart is presented in Fig. 3. The cc-pVQZ results confirm the tendency already observed with the cc-pVTZ basis set: the presence of the barrier is a consequence of the internal contraction while uc-MR-AQCC predicts no barrier and no van der Waals minimum. The uc- and ic-MR-AQCC data are plotted with the energy reference point at $r_2 = 15$ a.u. (see Fig. 3). This shows that the shape of the curves agrees for $r_2 > 6$ a.u. to better than 10 cm^{-1}, while for shorter distances they differ by about 900 cm^{-1} at $r_2 = 3.4$ a.u..

Table 7: Comparison of the fitted relative energies of the 1D - Cuts calculated with ic-MR-AQCC and MR-AQCC

1 D -Cut	E_0 (cm^{-1})	F_{xx}	F_{xxx}	F_{xxxx}	x_0 (a.u.ldeg)	N_{points}	$N_{parameters}$	rms (cm^{-1})
Barrier Bending Cut								
ic-MR-AQCC	-8.53	2.77276	-0.03502	0.00022	115.217	6	5	0.13
MR-AQCC	-10.37	2.85323	-0.03551	0.00024	115.062	6	5	0.15
Drift (%)	-17.71	-2.82	-1.39	-7.61	0.13			
Barrier Stretch Cut								
ic-MR-AQCC	-11.98	79339	-104652	89005	2.28719	8	7	0.00042
MR-AQCC	-6.44	78125	-102519	89807	2.29086	8	7	0.03091
Drift (%)	85.93	1.55	2.08	-0.89	-0.16			
Van-der-Waals Minimum Bending Cut								
ic-MR-AQCC	0.01	0.25597	-0.00340	0.00003	116.855	6	5	0.03
MR-AQCC	-0.02	0.27414	-0.00337	0.00003	116.641	6	5	0.02
Drift (%)	-158.00	-6.63	0.83	14.89	0.18			
O3 Stretch Cut								
ic-MR-AQCC	-30.28	39881	-60809	55286	2.42750	12	11	2.25
MR-AQCC	-52.75	38527	-58512	53675	2.43642	12	11	2.43
Drift (%)	-42.60	3.51	3.93	3.00	-0.37			
Dissociation Valence Cut								
ic-MR-AQCC	-1.957	80920	-106348	93291	2.29506	6	5	0.005
MR-AQCC	0.002	79930	-104494	92399	2.30005	6	5	0.004
Drift (%)	-89352.09	1.24	1.77	0.97	-0.22			
C_{2v} **Symmetric Cut**								
ic-MR-AQCC	-50.38	102880	-138070	124302	2.42181	9	8	0.00104
MR-AQCC	-85.59	100911	-134374	121174	2.42857	9	8	0.00589
Drift (%)	-41.14	1.95	2.75	2.58	-0.28			
C_{2v} **Bending Points**								
ic-MR-AQCC	-0.38	16.58420	-0.17207	0.00226	116.860	22	8	0.11
MR-AQCC	-0.25	16.55460	-0.17271	0.00225	116.897	22	8	0.15
Drift (%)	50.19	0.18	-0.37	0.65	-0.03			

Table 8: Comparison of the dissociation energies calculated with ic-MR-AQCC and MR-AQCC

basis set	ic-MR-AQCC (eV)	MR-AQCC (eV)
cc-pVTZ	0.887	0.973
cc-pVQZ	1.022	1.111
cc-pV5Z	1.078	1.173
cc-pV6Z	1.099	
(Q,5)	1.129	1.231
(5,6)	1.126	

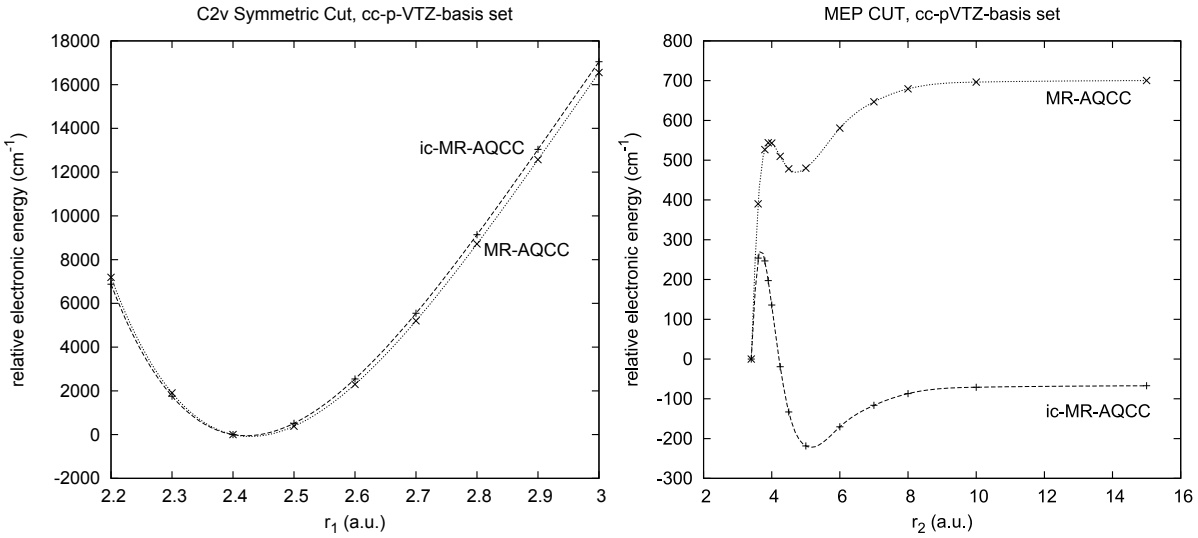


Figure 2: C_{2v} Symmetric Cut and MEP-Cut were calculated with MR-AQCC and ic-MR-AQCC and a cc-pVTZ-basis set. The energy within the C_{2v} Symmetric Cuts is relative to the ozone energy at $r_1 = 2.4$ a.u., $r_2 = 2.4$ a.u. and $\alpha = 117°$, within the MEP Cut relative to $r_1 = 2.275$ a.u., $r_2 = 3.4$ a.u. and $\alpha = 117°$.

The dissociation energy predicted at uc-MR-AQCC level is larger by 86 meV to 95 meV than its internally contracted counterpart (see Table 8). In fact, uc-MR-AQCC dissociation energy with a cc-pV5Z even overestimates the experimental value by 0.030 eV.
The results suggest, that the barrier is an artifact of the internal contraction which is also in line with the experimental finding of mass-independent isotope enrichment effect in ozone. In addition, this may indicate that the good agreement of the ic-MR-AQCC dissociation energy at CBS limit 1.126 eV and the experimental value 1.143 eV is owed to error compensation. On the other hand, overestimating the experimental dissociation energy at the CBS limit with the uc-MR-AQCC by 88 meV is larger than to be expected at this level of theory.

3.3 (ic)-MR-CISD

The energies calculated with MR-CISD+Q$_P$ and ic-MR-CISD+Q$_P$ almost reproduce the respective MR-AQCC data (Fig. 4).
The a posteriori Pople size consistency correction decreases the barrier height, lowers the van der Waals minimum and shifts the barrier maximum below the dissociation limit.

Also the MR-CISD energies a posteriorily Pople corrected are in agreement with the observed trends for MR-AQCC: the dissociation energies derived from the internally contracted variants only slightly underestimate the experimental value and the size-extensivity correction amounts to 0.06 eV at the CBS limit. The uncontracted calculations yield at cc-pV5Z level a higher dissociation energy by 0.062 eV (MR-CISD) compared to the ic calculation, a larger with the Pople correction (0.083 eV) also consistently leading to an overestimate of the experimental dissociation energies as compared to uc-MR-AQCC.Within the framework of the ic-MR-CISD method it is possible to increasingly better approximate the uc-MR-CISD method by increasing the number of reference states. To investigate the correctness of the uncontracted methods, the energies of $r_2 = 3.9$ a.u. of the MEP (barrier top)

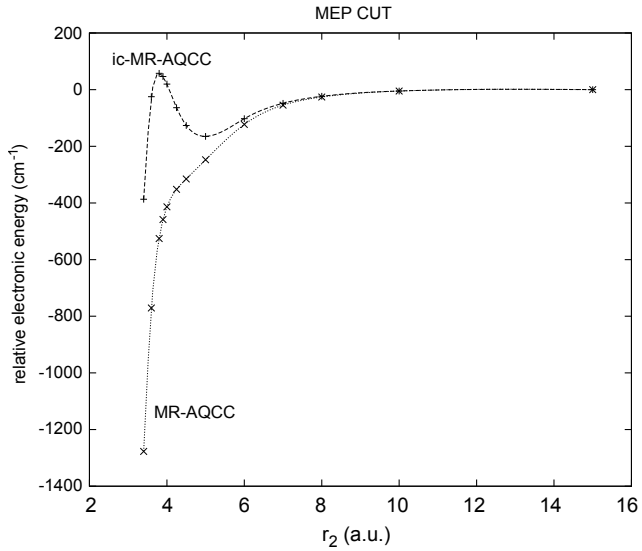


Figure 3: MEP-Cut calculated with MR-AQCC and ic-MR-AQCC and a cc-pVQZ-basis set. The energies of the MEP Cut are relative to ozone at $r_1 = 2.275$ a.u., $r_2 = 15$ a.u. and $\alpha = 117°$.

and $r_2 = 5.0$ a.u. of the MEP (van der Waals minimum) were calculated with the ic-MR-CISD+Q_P-method, a cc-pVTZ basis set and the application of more than one reference wave function (see Table 11), since the barrier should also decrease with the application of this method and more accounted electron correlation. The energy of the former barrier top decreases more with the number of the reference functions than the energy of the van der Waals minimum. In short, this tendency fits to the disappearing barrier with the MR-AQCC-method and MR-CISD+Q_P/+Q_D.

3.4 Calculation of Basis-set-superposition-error (BSSE)

The BSSE error was investigated at ic-MR-AQCC level only, because the BSSE of the ic-MR-AQCC method is expected to be similar to the BSSE uc-MR-AQCC as well as ic-MR-CISD+Q/uc-MR-CISD+Q with the same basis set.
The BSSE is approximately corrected for by the counter poise method which takes into account the distance dependence of the fragment energies in the presence of the respective ghost basis of the oxygen atom (O^*) and molecule (O_2^*):

$$\Delta E_{BSSE} = E_{O_2+O^*}(R_\infty) + E_{O+O_2^*}(R_\infty) - E_{O_2+O^*}(R) - E_{O+O_2^*}(R). \tag{15}$$

ΔE_{BSSE} is the BSSE correction. The infinite distance is approximated by $r_2 = 15$ a.u.; the bond length of O_2 is chosen $r_1 = 2.275$ a.u. for the MEP-Cut and $r_1 = 2.4$ a.u. for the BSSE correction of the dissociation energy.
At cc-pV6Z-basis, the calculated BSSE correction for ic-MR-AQCC is displayed in Fig. 5. This illustration implies that the dissociation energy corrected with the BSSE should be lower (see Tab. 12). The extrapolation of the BSSE correction was done with Equation 12. The BSSE correction energy was inserted into E_{Corr}.
Applying the correction calculated with ic-MR-AQCC to the MR-AQCC dissociation energy leads to a

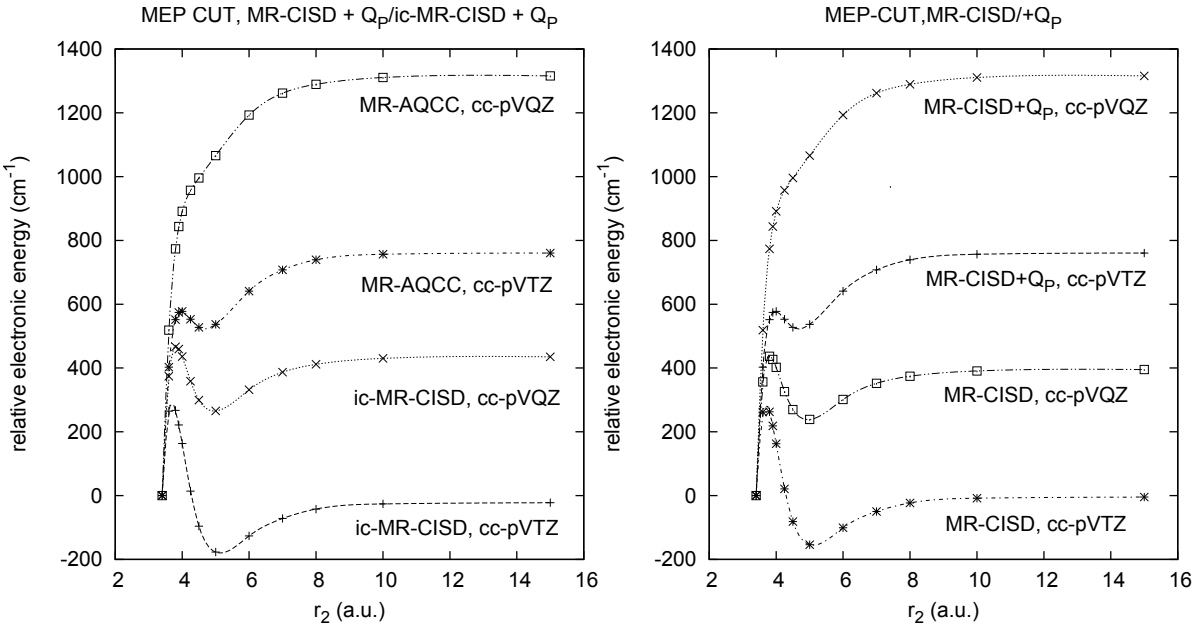


Figure 4: MEP-Cut was calculated with MR-CISD+Q_P and ic-MR-CISD+Q_P, and with MR-CISD and MR-CISD+Q_P. The energies of the MEP Cut are relative to ozone at $r_1 = 2.275$ a.u., $r_2 = 3.4$ a.u. and $\alpha = 117°$.

Table 9: Comparison of the dissociation energies calculated with ic-MR-CISD methods with and without size consistency correction

basis set	ic-MR-CISD (eV)	ic-MR-CISD+Q_D (eV)	ic-MR-CISD+Q_P (eV)	ic-MR-AQCC (eV)
cc-pVTZ	0.857	0.895	0.893	0.887
cc-pVQZ	0.982	1.032	1.030	1.022
cc-pV5Z	1.032	1.089	1.087	1.078
cc-pV6Z	1.051	1.111	1.109	1.099
(Q,5)	1.077	1.142	1.139	1.129
(5,6)	1.075	1.138	1.136	1.126

Table 10: Comparison of the dissociation energies calculated with MR-CISD methods with and wihtout size consistency correction

basis set	MR-CISD (eV)	MR-CISD+Q_D (eV)	MR-CISD+Q_P (eV)	MR-AQCC (eV)
cc-pVTZ	0.916	0.984	0.983	0.973
cc-pVQZ	1.040	1.124	1.123	1.111
cc-pV5Z	1.094	1.191	1.177	1.173
(Q,5)	1.140	1.235	1.234	1.231

Table 11: Relative energies of the barrier top and the van der Waals minimum calculated with ic-MR-CISD+Q_P, a cc-VTZ basis set and a number of reference wave functions and compared with the MR-CISD+Q_P-energy

Number of reference wave functions	Relative energies (cm^{-1}) barrier top $r_2 = 3.9$ a.u.	van der Waals minimum $r_2 = 5.0$ a.u.
1	0	0
2	-75.11	-52.52
4	-133.11	-82.83
5	-143.14	-89.35
10	-191.73	-108.36
MR-CISD+Q_P	-3175.07	-2813.97

$D_e = 1.148\,\mathrm{eV}$ at the cc-pV5Z level, thereby overestimating the dissociation energy by $0.005\,\mathrm{eV}$. Thus, the BSSE cannot account for the discrepancy with the experimental value, since the dissociation energy at a cc-pV6Z basis set will be larger while the BSSE decreases with increasing basis set size. Moreover, in the limit of an infinite basis set the BSSE should vanish (see Table 12).

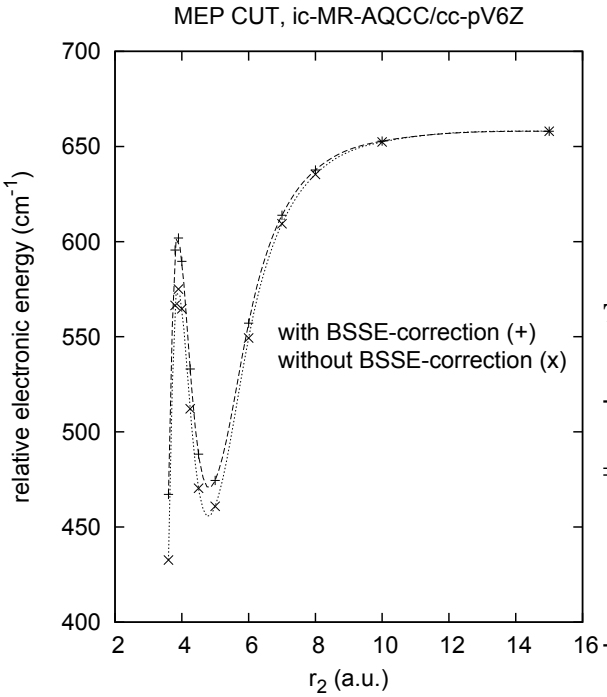


Figure 5: Influence of the BSSE to the Minimum Energy Path (ic-MR-AQCC/cc-pV6Z). The energies with BSSE correction (+) and without BSSE correction (x) are relative to the uncorrected ozone energy at $r_1 = 2.275$ a.u., $r_2 = 3.4$ a.u. and $\alpha = 117°$.

Table 12: Correction of the dissociation energy regarding the basis set superposition error (ic-MR-AQCC-method)

basis set	corrected dissociation energy (eV)	dissociation energy (eV)
cc-pVTZ	0.459	0.887
cc-pVQZ	0.967	1.022
cc-pV5Z	1.053	1.078
cc-pV6Z	1.087	1.099
(Q,5)	1.124	1.129
(5,6)	1.122	1.126

4 Summary

The minimum energy path calculated with ic-MR-AQCC and ic-MR-CISD+Q_P show the typical barrier and van der Waals minimum as in former calculations. The experimental dissociation energy for the ozone dissociation ($O_3(^1A_1) \longrightarrow O(^3P) + O_2(^3\Sigma_g^-)$) is underestimated by $0.021\,\mathrm{eV}$ (ic-MR-AQCC) and $0.011\,\mathrm{eV}$ (ic-MR-CISD+Q_P) including BSSE corrections of $0.004\,\mathrm{eV}$. This good agreement with the experimental value is likely to be due to error compensation, since the conceptually more accurate uncontracted variants, MR-AQCC and MR-CISD+Q_D/Q_P, overestimate the experimental dissociation energies substantially already at the cc-pV5Z level. Furthermore, the MEP cuts computed with MR-AQCC and MR-CISD+Q_D/Q_P do not display a barrier which is in line with the observed isotope enrichment effect. The discrepancy for the dissociation energy using uc-MR-AQCC and uc-MR-CISD+Q_D/Q_P-methods is not related to the basis set superposition error. Since the dissociated

fragments in contrast to the ozone molecule constitute highly symmetrical species, the deviation might arise from some artifical bias in the choice of the configuration space.

5 Acknowledgements

I would like to thank my superviser Dr. T. Müller for his support with advice and knowledge. Moreover, I am very grateful for the recommendation of Prof. R. Dronskowski, the suggestion of this guest student program by my former superviser Dr. M. Lumeij and the tips of the former guest student Stefan Maintz. Last but not least, I would like to express my gratitude to Mathias Winkel and Natalie Schröder for their organization of the guest student program and their support during the stay.

References

1. M. RUBIN, *Bull. Hist. Chem* **26(1)**, 40 (2001).
2. Y. GAO and R. MARCUS, *Science* **293**, 259 (2001).
3. F. HOLKA, P. G. SZALAY, T. MÜLLER, and V. G. TYUTEREV, *J. Phys. Chem. A* **114**, 9927 (2010).
4. V. TYUTEREV, S. TASHKUN, P. JENSEN, A. BARBE, and T. COURS, *J. Mol. Spectrosc.* **198**, 57 (1999).
5. V. TYUTEREV, S. TASHKUN, D. SCHWENKE, P. JENSEN, T. COURS, A. BARBE, and M. JACON, *Chem. Phys. Lett.* **316**, 271 (2000).
6. E. SCHRÖDINGER, *Ann.Phys.* **384**, 361 (1926).
7. L. GONZALEZ and D. BENDER, *Einführung in die computergestützte Quantenchemie*, pp. 391–413, eXamen.press, Springer Berlin Heidelberg, 2010.
8. W. PAULI, *Z. Phys. A-Hadron Nucl.* **31**, 765 (1925).
9. C. C. J. ROOTHAAN, *Rev. Mod. Phys.* **23**, 69 (1951).
10. G. G. HALL, *Proc. R. Soc. A* **205**, 541 (1951).
11. F. JENSEN, *Introduction to computational chemistry*, John Wiley & Sons, 2007.
12. J. REINHOLD, *Quantentheorie der Moleküle: Eine Einführung*, Teubner Studienbücher, Teubner, 2006.
13. I. SHAVITT, *in "Methods of Electronic Structure Theory"*, pp. 189-275, Ed. H. F. Schaefer III, Plenum, New York, 1977.
14. P. G. SZALAY and R. J. BARTLETT, *Chem. Phys. Lett.* **214**, 481 (1993).
15. T. H. DUNNING JR., *J. Chem. Phys.* **90**, 1007 (1989).
16. T. H. DUNNING, JR., *J. Chem. Phys.* **90**, 1007 (1989).
17. A. WILSON, T. VAN MOURIK and T. H. DUNNING, JR., *J. Mol. Struct. (THEOCHEM)* **388**, 339 (1996).
18. X. W. SHENG, L. MENTEL, O. V. GRITSENKO, and E. J. BAERENDS, *J. Comp. Chem.* **32**, 2896 (2011).
19. A. HALKIER, T. HELGAKER, P. JORGENSEN, W. KLOPPER, H. KOCH, J. OLSEN, and A. K. WILSON, *Chem. Phys. Lett.* **286**, 243 (1998).
20. H.-J. WERNER and P. J. KNOWLES, *Theor. Chim Acta* **78**, 175 (1990).
21. H.-J. WERNER and P. KNOWLES, *J. Chem. Phys.* **89**, 5803 (1988).
22. P. KNOWLES and H.-J. WERNER, *Chem. Phys. Lett* **145**, 514 (1988).
23. H.-J. WERNER, P. J. KNOWLES, G. KNIZIA, F. R. MANBY, M. SCHÜTZ, et al., MOLPRO, version 2010.1, a package of ab initio programs, 2010.
24. H.-J. WERNER and P. J. KNOWLES, *J. Chem. Phys* **82**, 5053 (1985).
25. P. J. KNOWLES and H.-J. WERNER, *Chem. Phys. Lett.* **115**, 259 (1985).
26. S. R. LANGHOFF and E. R. DAVIDSON, *Int. J. Quantum Chem.* **8**, 61 (1974).
27. E. R. DAVIDSON, *The World of Quantum Chemistry* (1974), in: R. Daudel, B. Pullman, Editors Reidel, Dordrecht.
28. J. A. POPLE, R. SEEGER, and R. KRISHNAN, *Int. J. Quantum Chem.* **S11**, 149 (1977).
29. H. LISCHKA, R. SHEPARD, F. B. BROWN, and I. SHAVITT, *Int. J. Quantum Chem., Quantum Chem. Symp.,* **15**, 91 (1981).
30. R. SHEPARD, I. SHAVITT, R. M. PITZER, D. C. COMEAU, M. PEPPER, H. LISCHKA, P. G. SZALAY, R. AHLRICHS, F. B. BROWN, and J. ZHAO, *Int. J. Quantum Chem., Quantum Chem. Symp.,* **22**, 149 (1988).

31. H. LISCHKA, R. SHEPARD, R. M. PITZER, I. SHAVITT, M. DALLOS, T. MÜLLER, P. G. SZALAY, M. SETH, G. S. KEDZIORA, S. YABUSHITA, and Z. ZHANG, *Phys. Chem. Chem. Phys.* **3**, 664 (2001).
32. H. LISCHKA, R. SHEPARD, I. SHAVITT, R. M. PITZER, M. DALLOS, T. MÜLLER, P. G. SZALAY, F. B. BROWN, R. AHLRICHS, H. J. BÖHM, A. CHANG, D. C. COMEAU, R. GDANITZ, H. DACHSEL, C. EHRHARDT, M. ERNZERHOF, P. HÖCHTL, S. IRLE, G. KEDZIORA, T. KOVAR, V. PARASUK, M. J. M. PEPPER, P. SCHARF, H. SCHIFFER, M. SCHINDLER, M. SCHÜLER, M. SETH, E. A. STAHLBERG, J.-G. ZHAO, S. YABUSHITA, Z. ZHANG, M. BARBATTI, S. MATSIKA, M. SCHUURMANN, D. R. YARKONY, S. R. BROZELL, E. V. BECK, , J.-P. BLAUDEAU, M. RUCKENBAUER, B. SELLNER, F. PLASSER, and J. J. SZYMCZAK, COLUMBUS, an ab initio electronic structure program, release 7.0, 2010.

Evaluation of preconditioners for large sparse matrices

Alexander Alperovich

The Blavatnik School of Computer Science
Raymond and Beverly Sackler faculty of exact sciences
Tel-Aviv University
Tel-Aviv, Israel

E-mail: sashaa@post.tau.ac.il

Abstract:
In this report I present the outcome of an examination and benchmark of several software packages for solving linear systems of equations using various preconditioners. The preconditiones examined are based on the Algebraic Multigrid method, Incomplete LU decomposition and the Frobenius Norm approximation. The packages use Krylov-based iterative methods such as Restarted Generalized Minimum Residual (GMRES) or Conjugate Gradient (CG) as the solvers, allowing different approaches to be set as the preconditioner.
The benchmark was performed on the supercomputers of the JSC, JUGENE and JUROPA.

---

1 Introduction

Given a linear system of equations $A^{n\times n} \cdot b = x$ one can apply some well known methods, such as Gaussian elimination, in order to solve the system. Using such a method has some major drawbacks that make them impractical for nowadays systems that grow larger and larger. The asymptotic time complexity of these methods is $O(n^3)$ means doubling the problem domain leads to 8 times longer computation, a rate that is not provided by the growth of the available hardware.
A drawback of using Gaussian ellimination for sparse systems is the loss of sparsity during the process which leads to additional memory consumption up to the level that current computer systems might not be able to provide.
These two issues, among others, lead to a quest for a faster and sparsity preserving methods - preconditioners. The preconditioner is a matrix P that minimizes the condition number of AP^{-1}, thus allowing a faster solution of $A \cdot P^{-1}y = b$ which is later followed by $P \cdot x = y$ giving us the final result.
The seek for a perfect preconditioner constructing technique - such that will fit all the possible matrices - is a very old and well researched. Still, no such technique is known and one must decide what is the appropriate way to construct a perconditioner for a given problem. Many software packages exist that provide the user with a preconditioner for an input, utilizing methods that are based on numerical analysis, heuristics, similarity and more.
In this work two software packages - Hypre [1] and ILUPack [2] were examined on the supercomputers

of the JSC. The construction of the preconditioners of Hypre have a parallel implementation and thus allow utilizing the supercomputer and distributing the workload among the multiple nodes using MPI. The rest of this report is organized as follows. First, Section 2 presents the background for this work. Section 3 presents the software packages that were examined during this work. Section 4 contains the description of the input matrices that were used for this work. The results of the work are presented in Section 5. Our conclusions from this study are presented in Section 6.

2 Background

2.1 Krylov subspace

The $n \times m$ Krylov matrix generated by matrix $A^{n \times n}$ and vector b is defined as

$$\mathcal{K}_n = \begin{bmatrix} | & | & | & | & & | \\ b & Ab & A^2b & \cdots & & A^{m-1}b \\ | & | & | & | & & | \end{bmatrix}$$

This matrix is a starting point for some of the most successful methods in numerical linear algebra for finding (some) eigenvalues of a given matrix - Lanczos, Arnoldi and more, and for solving large sparse systems of linear equations - GMRES, Conjugate gradient and others. As constructing each additional vector in the subspace requires only a single matrix-vector product, some black box method for performing this computation can be utilized, be it a sparse multiplication, parallel one or other. The dimension of the subspace spanned by the vectors generated is m and the vectors

$$\{A^k \cdot b\} \qquad k = 0 \cdots m-1$$

are the base of the subspace. However, this base is suboptimal for this subspace, and the generated vectors are nearly linearly dependent, therefore one should use some orthogonalization scheme in order achieve better results.

2.2 Generalized Minimum Residual

The GMRES [3] method is an iterative method based on the constructed Krylov subspace and finding the vector that minimizes the Euclidean norm of the residual using the Arnoldi iteration. The method works on arbitrary non singular square matrices, by constructing the appropriate Krylov subspace, and for each iteration finds the vector $x_n \in \mathcal{K}_n$ that minimizes

$$\|r_2\| = \|b - Ax_n\|_2$$

The minimization is performed by using the Arnoldi iteration in order to generate an orthonormal basis $q_1\, q_2 \ldots q_n$ that form the matrix Q_n, so we seek for the vector y_n such that $x_n = Q_n \cdot y_n$. An

additional outcome of the Arnoldi iteration is the upper $(n+1) \times n$ Hessenberg matrix $\tilde{H_n}$ such that

$$AQ_n = Q_{n+1}\tilde{H_n}$$

Next we deduce $\|Ax_n - b\| = \tilde{H_n}y_n - \beta e_1$ with $\beta = \|b - Ax_0\|$, e_n is the first vector in the standard basis of R^{n+1} and x_0 is the initial guess. This allows us to solve the least square problem of size n

$$r_n = \tilde{H_n}y_n - \beta e_1$$

The GMRES algorithm can be expressed as follows:
while the residual is bigger than the treshold do

1. do one step of Arnoldi iteration
2. find the y_n which minimizes $\|r_n\|$
3. $x_n = Q_n y_n$

end do

The complexity of the algorithm depends on the black box matrix vector multiplication operation. For a sparse matrix that have nnz non zero elements, this operation can be easily done within $O(nnz)$ floating point operations. In addition to vector matrix multiplication, at every iteration the minimizing vector must be sought for, this taking $O(mn)$ floating point operations for the m-th iteration.

2.2.1 Restarted Generalized Minimum Residual

The restarted GMRES or GMRES(k) is a method to overcome the quadratic growth of the iteration cost due to the $O(mn)$ component for the m-th iteration. Instead of keeping the old basis starting x_0 for the whole process, it is restarted after k steps having x_k as the initial guess. This results in better asymptotic complexity but might harm convergence in case k is not big enough to allow constructing such $\mathcal{K}_n$ that allows minimizing x_n.

3 Software packages

This work is based on utilizing a few software packages dealing with solving systems of linear equations through usage of preconditioned GMRES.

3.1 Hypre

The Hypre project [1] aims to provide a software package that deals with large sparse linear systems of equations utilizing parallel and distributed computer systems. The package allows choosing and setting up both a solver and a preconditioner from a large set of implemented algorithms according

to the data representation used - the conceptual interface. The interfaces used in Hypre are as following:

- Structured-Grid System Interface (Struct): This interface is appropriate for applications whose grids consist of unions of logically rectangular grids with a fixed stencil pattern of nonzeros at each grid point. This interface supports only a single unknown per grid point.
- Semi-Structured-Grid System Interface (SStruct): This interface is appropriate for applications whose grids are mostly structured, but with some unstructured features. Examples include block-structured grids, composite grids in structured adaptive mesh refinement (AMR) applications, and over-set grids. This interface supports multiple unknowns per cell.
- Finite Element Interface (FEI): This is appropriate for users who form their linear systems from a finite element discretization. The interface mirrors typical finite element data structures, including element stiffness matrices.
- Linear-Algebraic System Interface (IJ): This is the traditional linear-algebraic interface. General solvers and preconditioners are available through this interface

After choosing the conceptual interface one has to choose the solver and the preconditioner to use from the following list. Some of the algorithms can be used only with appropriate conceptual interfaces, and few can be used as a solver, a preconditioner or both.

Jacobi	SMG	PFMG	Split	SysPFMG
FAC	Maxwell	BoomerAMG	AMS	MLI
ParaSails	Euclid	PILUT	PCG	GMRES
FlexGMRES	LGMRES	BiCGSTAB	Hybrid	LOBPCG

In this work 3 of the preconditioners of the package were examined through the IJ interface using the GMRES solver. Following is some overview of the preconditioners used in this project.

3.1.1 AMG

BoomerAMG is a parallel implementation of the algebraic multigrid method. The user can choose between various different parallel coarsening techniques, interpolation and relaxation schemes through parameter interface for the package.
The AMG method is based on approximating a solution for a coarse problem and then interpolating the result to a finer grid. The method first deals with a high frequency errors thus pushing the error down at the global scale. This method is known to perform well on differential equations but can be used as a preconditioner for linear systems of equations as well.

3.1.2 Euclid

The Euclid preconditioning framework encapsulates a few techniques of preconditioner construction. The current version of Hypre includes the parallel ILU(k) and the Block Jacobi ILU(k) algorithms. These preconditioners are achieved by performing an incomplete LDU decomposition of the original

matrix, thus reaching a sparse forms that still allows a fast deduction of determinants and thus solving the system. As the process of LDU decomposition involves multiple communication and synchronization acts among the processors participating at the parallel algorithm it is not recomended to use this method for most of the problems when utilizing just a few processors as the overhead will be greater than the benefit of using more than a single core.

3.1.3 Parasails

ParaSails is a parallel implementation of a sparse approximate inverse preconditioner, using a priori sparsity patterns and least-squares (Frobenius norm) minimization. Symmetric positive definite (SPD) problems are handled using a factored SPD sparse approximate inverse. General (nonsymmetric and/or indefinite) problems are handled with an unfactored sparse approximate inverse. It is also possible to precondition non-symmetric but definite matrices with a factored, SPD preconditioner. ParaSails uses a priori sparsity patterns that are patterns of powers of sparsified matrices. ParaSails also uses a post-filtering technique to reduce the cost of applying the preconditioner.

3.2 ILUPack

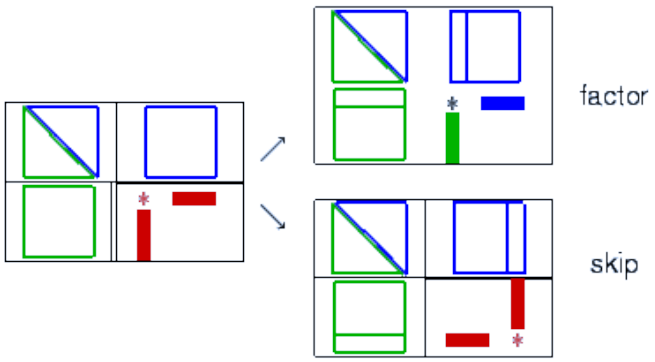
ILUPack states its goal as numerical solution of large sparse linear systems $Ax = b$ by

- Iterative methods, in particular preconditioned Krylov subspace methods
- Preconditioner constructed from an incomplete LU decomposition

The solvers it provides are the GMRES(k) for arbitrary matrices and CG for symmetric positive definite matrices.
The preconditioners are constructed by performing nested ILDU decomposition of the matrix, the nested decomposition is presented in the following 2 figures. First - the decomposition itself to 4 components.

$$A = \begin{pmatrix} B & F \\ E & C \end{pmatrix} = LDU + E \approx$$  , where L, U are unit diagonal.

The frameworks performs the decomposition as long as it consider it reasonable, and then recursively applies the same technique to the unfactored components as long as some treshold is kept.



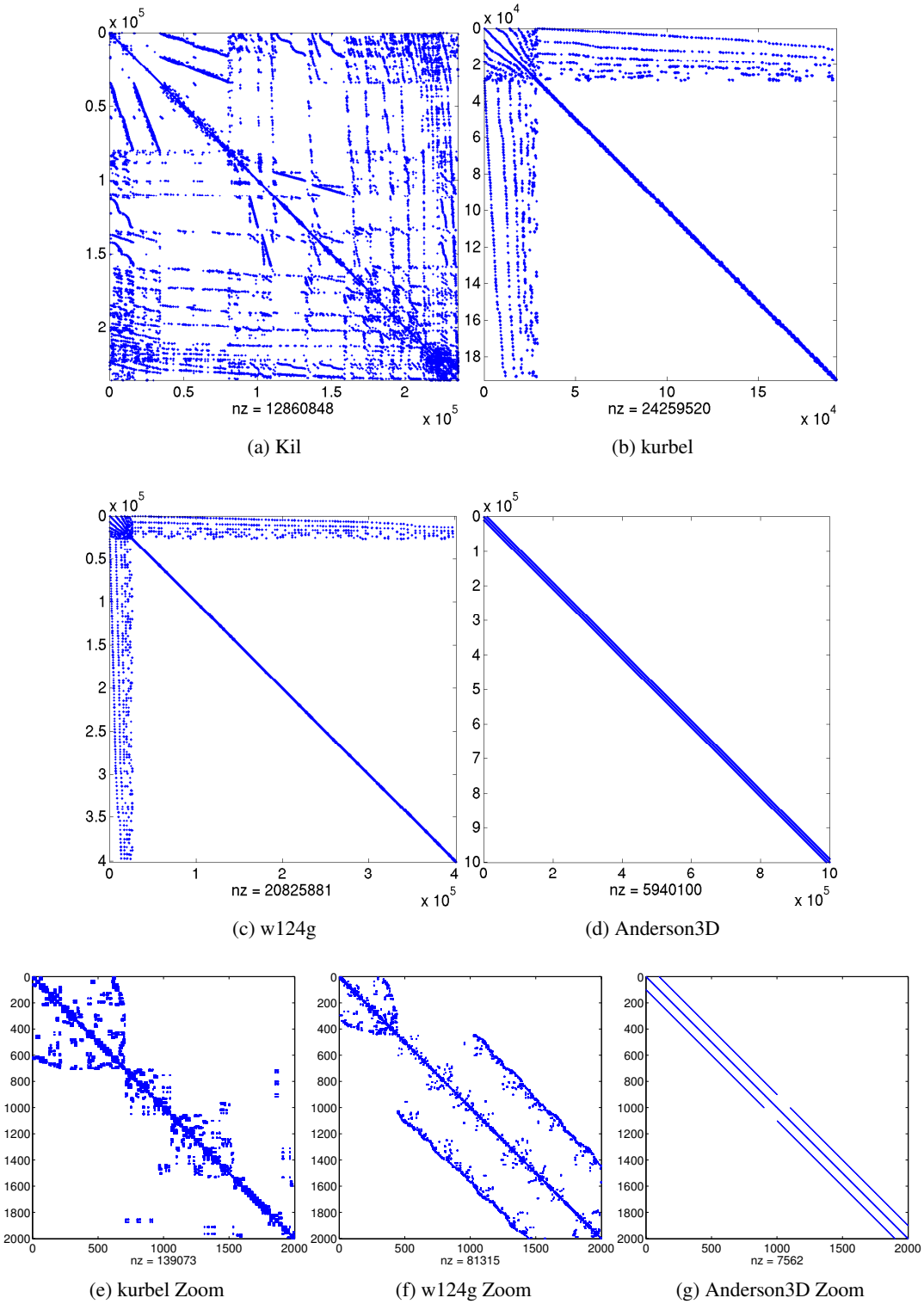
4 Data

This section presents the input matrices used for the benchmark. All these matrices are known to make a hard life to the various solvers, they are very sparse and relatively large.
For testing we used 4 matrices, all of them represent 3D connectivity for various structures. Table 1 presents the characteristics, such as the dimension, the nnz, whether the matrix is symmetric positive definite and gives some general description regarding the matrix domain.
During the benchmark all the matrices besides Anderson3D were presented in the binary CRS form and occupied $(150 - 300)MB$ on disk. The Anderson3D matrix was generated on the fly by the software according to its pattern.

Matrix	Dimension	nnz	SPD	Description
Kil	$235,962 \times 235,962$	$12,860,848$	YES	Related to shell structure (pressed metal)
kurbel	$192,858 \times 192,858$	$24,259,520$	YES	Related to the 3D structure of an engine crankshaft
w124g	$401,595 \times 401,595$	$20,825,881$	YES	Related to 3D steel structure
Anderson3D	$1,000,000 \times 1,000,000$	$5,940,100$	NO	Related to the (quantum) theory of semiconductors

Table 1: Matrices characteristics

One of the most important details regarding the input matrix is its sparsity pattern. The ability to distribute the workload evenly among the processors, the convergence factor, initial condition numbers - are all defined by the pattern. Figure 1 presents the sparsity patterns of the 4 input matrices. The bottom row zooms at the center of the matrix with radius of 1000. One can notice that the matrices are extremely sparse with $nnz \in [5n, 126n]$.



(a) Kil (b) kurbel

(c) w124g (d) Anderson3D

(e) kurbel Zoom (f) w124g Zoom (g) Anderson3D Zoom

Figure 1: Sparsity patterns of test matrices

5 Results

Here the results of the benchmark are presented. For all the experiments the GMRES solver was used, with different restart parameters. First, Table 2 summarizes the result, presenting which matrices were successfully solved using which preconditioner and on which machine. We can see that the Hypre-AMG preconditioner was able to cope with Anderson3D on both Jugene and Juropa and with kurbel on Juropa only. Hypre-Euclid was successful only on Jugene with Kurbel. The Hypre-Parasails algorithm managed to cope only with Kil on Jugene. The ability of one machine to cope with a specific matrix while failing on other machine first seems weird, as the code executed is expected to be identical. The probable variations are due to the limitations of the machines - on Jugene one can only ask for a short computation time and much more limited amount of local memory thus allowing longer and more memory consuming computations to succeed on Juropa while failing on Jugene. On the other hand, Jugene can provide many more processors for the computation, and this may lead to convergence at places Juropa failed.

Finaly, the ILUPack was able to solve all the matrices, Anderson3D for any requested tolerance and the other ones to the tolerances presented at the table. Clearly, as the ILUPack comes compiled for the Intel architecture only, it could not be able to be examined on Jugene and the results presented are coming from Juropa only.

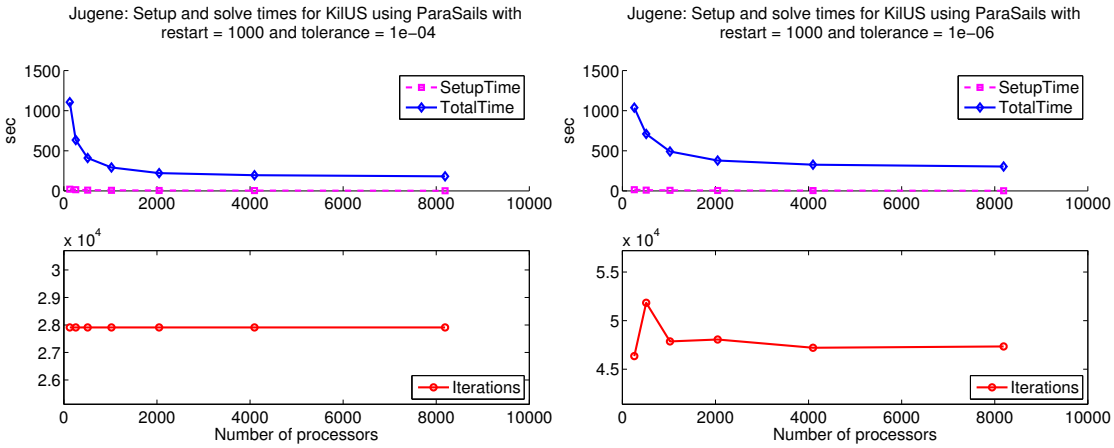
	Kil	Kurbel	w124g	Anderson3D
Hypre - AMG	X	Juropa	X	V
Hypre - Euclid	X	Jugene	X	X
Hypre - Parasails	Jugene	X	X	X
ILUPack	1e-5	1e-3	1e-3	V

Table 2: Results summary

5.1 Hypre

Next some more detailed results are to follow. We first address the Hypre package, and as it has a parallel implementation we first show the scalability in terms of time and the number of iterations for the Kil matrix on Jugene using the Parasails preconditioner. Figure 2 shows 2 plots for various terms of tolerance requirement for the solver. The x axis is the number of processors being involved in the computation.

One can clearly see the scalability of the method - and the asymptotic behavior that depends on the requested tolerance. As for the setup times - the Parasails method is easy to construct and, as it is possible to perform row distribution of the preconditioner matrix among the processors, it requires almost no synchronization or communication overhead. This leads to very short setup times as can be seen in the plots.
Another interesting aspect of the same problem is presented in Figure 3. Here the x axis is the tolerance parameter and one can see the behavior of the time and iteration count for a given number of



(a) $Tolerance = 1e^{-4}$ (b) $Tolerance = 1e^{-6}$

Figure 2: Times and iterations for Kil on Jugene using Parasails preconditioner

processors. As the convergence rate is nearly constant for a specific problem one can extrapolate these results to approximate the expected number of iterations and times for other various tolerance values.

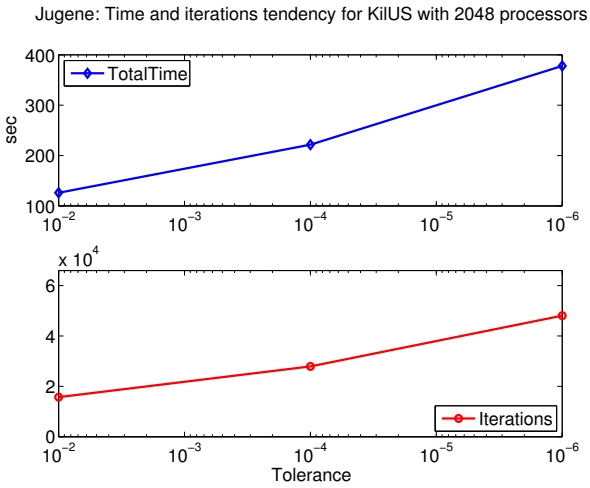
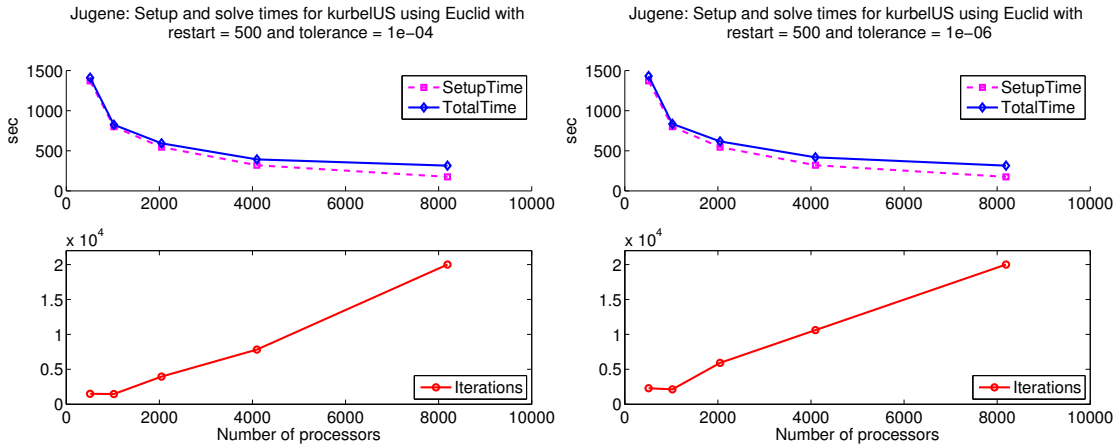


Figure 3: Times and iterations for kil on Jugene using Euclid Parasails with 2048 processors vs tolerance

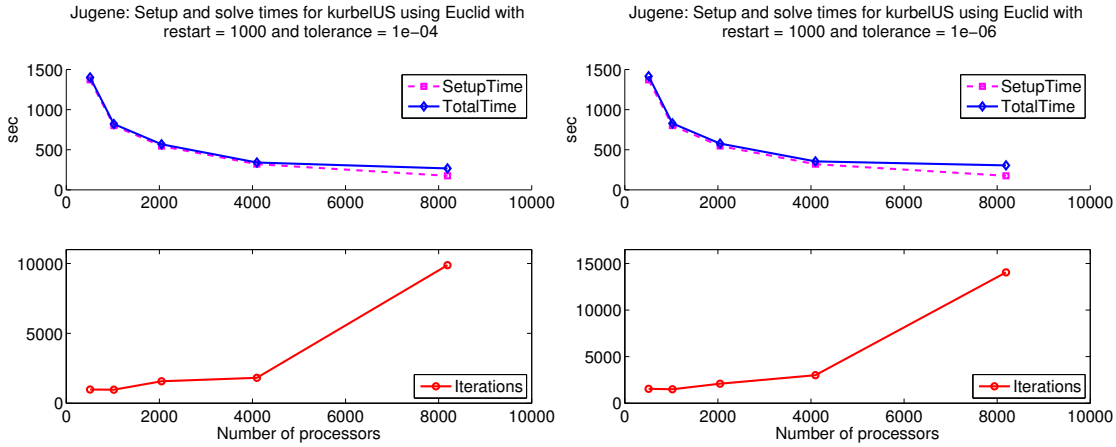
Next, we examine the kurbel matrix on Jugene using the Euclid preconditioner with 2 different restart options - $m = 500$ and $m = 1000$.

For the Euclid preconditioner algorithm one can clearly see that the major part of the time spent during the run is at the setup time. This is due to the construction of the preconditioner matrix which involves the ILU decomposition of the original matrix, thus requiring a lot of synchronization and communication between the processors. Still, once you can utilize a few thousands processors for the task there are results. The iteration count grows rapidly as the number of processors grow.
The restart parameter m sets the dimension of the Krylov subspace. It is clear that a greater dimension



(a) $Tolerance = 1e^{-4}$ (b) $Tolerance = 1e^{-6}$

Figure 4: Times and iterations for kurbel on Jugene using Euclid preconditioner with $m = 500$

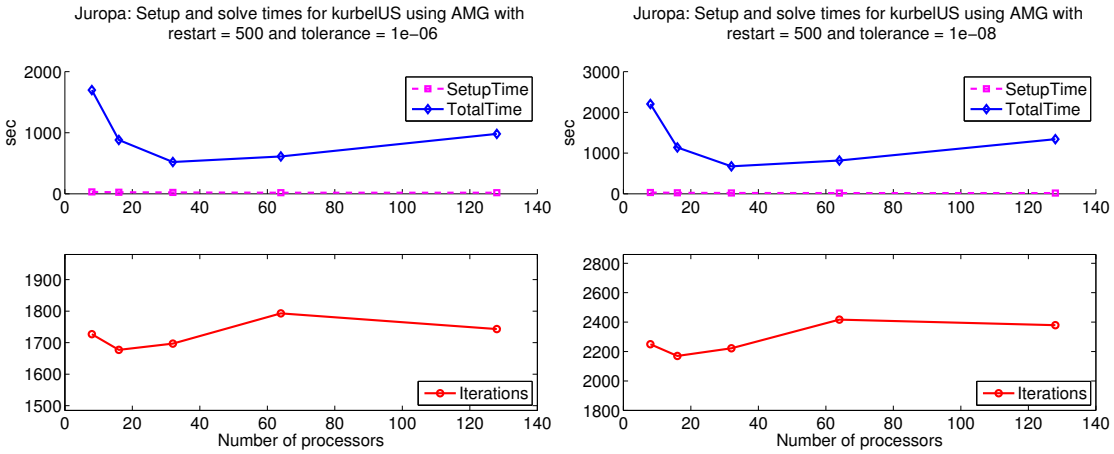


(a) $Tolerance = 1e^{-4}$ (b) $Tolerance = 1e^{-6}$

Figure 5: Times and iterations for kurbel on Jugene using Euclid preconditioner with $m = 1000$

may lead to convergence where a smaller one would not. Yet, this has a cost in time and memory complexity as the bigger basis must be constructed and stored and also regarded during the minimization phase. Comparison of Figure 4 and Figure 5 shows clearly that m has a major influence on the iteration count, as well as the time spent at the solve phase - the margin between the setup and the total times. Clearly, one must adjust this parameter to proper fit his problem domain.
Last, Figure 6 displays the behavior of the AMG preconditioner on kurbel matrix on the Juropa machine.

Unlike previous experiments on Jugene, here one can clearly see there is a minimal time that occurs when using 32 processors, and it grows when using more. The number of iterations when using AMG for this problem is more or less constant, and similar to Parasails the setup time is negligible.



(a) $Tolerance = 1e^{-6}$ (b) $Tolerance = 1e^{-8}$

Figure 6: Times and iterations for kurbel on Juropa using AMG preconditioner

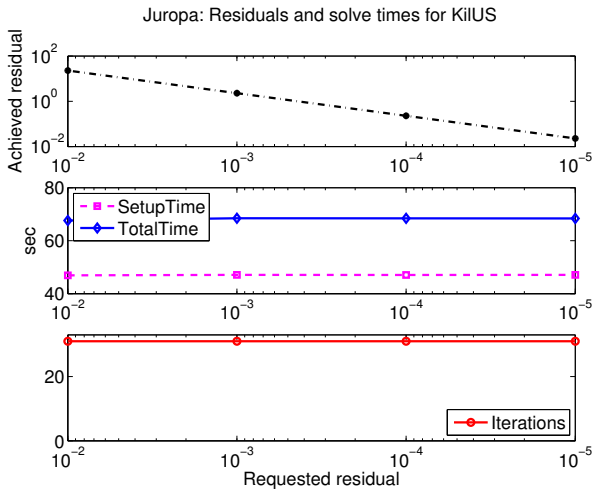
5.2 ILUPack

The ILUPack distributable contains a precompiled library for the Intel architecture, thus it could be only examined on Juropa. Moreover, the implementation is sequential, which means it can not utilize SMP or distributed environments and can only be run on a single processor. One rather important thing to notice regarding ILUPack is the soft threshold it provides - which means setting the requested residual will not eventually provide you with a solution up to that tolerance but rather some effort to reach it. The software determines itself when it should stop according to some heuristic computation it performs prior running the solver and not verifying the final residual versus the requested one, which leads to the appearance of a new outcome - the achieved residual vs. the requested one. Figure 7 present the result of executing the package on various matrices. The x axis for these plots is the requested residual.

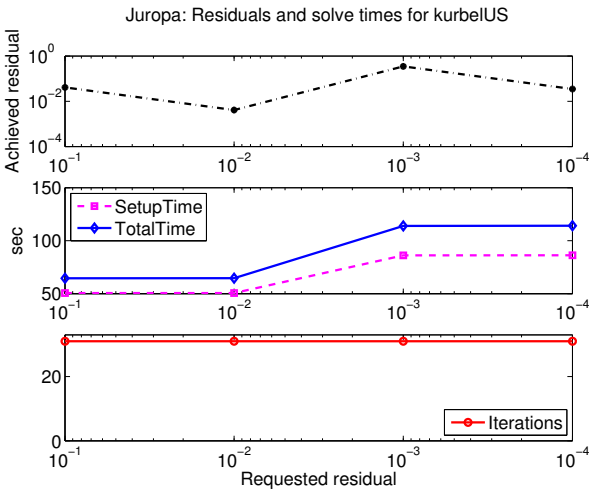
For the Kil matrix, the achieved residual decreases as the requested tolerance drops, a behavior that can be expected from the solver. This is also the case with the Anderson3D matrix. On the other hand, with the kurbel and w124g matrices this is not the case - at some points asking for better precision leads to a greater residual - a behavior that is unwanted for such a package.
Similar to the Euclidian preconditioner of the Hypre package, and due to the similarity of the methods, the major part of the time is spent on constructing the preconditiner, and this time seems to be more or less constant for these matrices. The iteration count is constant for all problems besides Anderson3D which require more iterations in order to achieve better convergence, which also lead to longer total time for the entire computation.

6 Conclusions

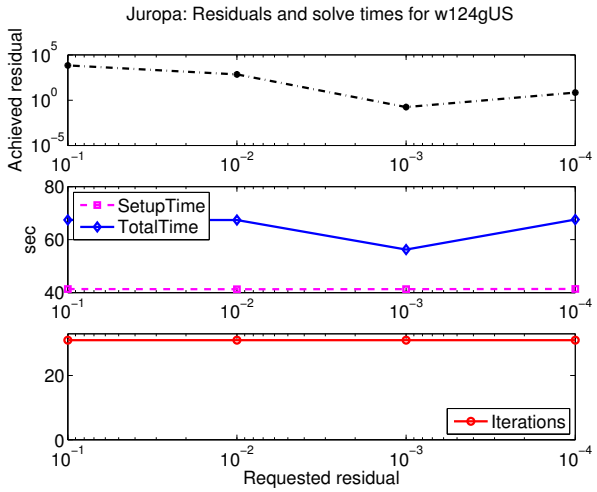
This work focused on examination of two software packages for solving linear systems of equations using iterative, Krylov subspace based preconditioners on the JSC supercomputers. The Hypre package



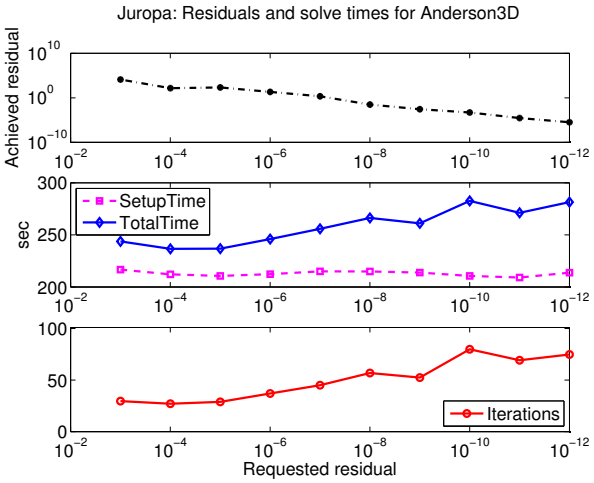
(a) Kil



(b) kurbel



(c) w124g



(d) Anderson3D

Figure 7: Achieved residual, times and iteration counts for the ILUPack on various matrices

has a parallel implementation and can be executed on both Jugene and Juropa. The ILUPack involves ILU decomposition - a step that is extremely costy for parallel implementation and so there is no parallel version of the package yet, and there is no version for IBM/BlueGene so it can be run only on Juropa.

Different packages and different input data as well as other parameters such as number of steps before restarting greatly infuence the outcome of the experiment. The scalability patterns as well as general package capabilities can help users decide which of the packages should they utilize for their work having structural similarity and resource availability in mind.

7 Acknowledgments

The author would like to thank his advisor - Dr. Bernhard Steffen for proposing and leading the project and his helpful insights, Mathias Winkel and Natalie Schröder for the organization of the program. I also want to thank Prof. Sivan Toledo from the Tel Aviv university for recommending me for this program, the other guest students for for making this a pleasant experience and my girlfriend - Dali Cohn - for coming with me.

References

1. https://computation.llnl.gov/casc/linear_solvers/sls_hypre.html
2. http://www-public.tu-bs.de/ bolle/ilupack/
3. Y. Saad and M. H. Schultz *GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems* SISSC 7, pages 856-869 (1986)

A Coulomb Solver Based on a Parallel NFFT for the ScaFaCoS Library

Sebastian Banert

Chemnitz University of Technology
Department of Mathematics
Reichenhainer Straße 39
09114 Chemnitz

E-mail: sebastian.banert@s2008.tu-chemnitz.de

Dedicated to Professor Klaus Banert on the occasion of his 56th birthday.

Abstract:
We describe a NFFT-accelerated Ewald summation for calculating electrostatic potentials and forces for a periodic system of charged particles. Furthermore, we have a glance at our implementation of this method within the ScaFaCoS library and present the results for some special systems. In the last part, we will give stimulations for further research and developments.

1 Introduction

The simulation of particles is an important toolkit for natural as well as material science. A particular case is the computation of Coulomb interactions between a system of charged particles. Important parameters in the simulation are potentials to estimate the total energy and thus the stability of certain configurations. On the other hand, the forces acting at each particle can be used to simulate motions of the system over a short time interval.

Since there are no analytic formulae for the motion of more than two mutually interacting particles, numerical simulation is an indispensable method for the analysis of those systems. In spite of the development of modern computer systems, it is necessary to have efficient algorithms to get realistic results for large and complicated configurations within a reasonable period of time.

Due to physical limitations, the efficiency of current processors is not rising any longer. To get more powerful computer systems anyway, it is necessary to run multiple processors in parallel. To benefit from these techniques, the algorithms have to be adapted to this kind of hardware.

2 Notation

We denote by $\mathbb{Z}, \mathbb{N}, \mathbb{R}, \mathbb{C}$ the set of all integers, positive integers, real numbers and complex numbers, respectively. The usual Euclidean space is $\mathbb{R}^3$, whose elements will be written in boldface, i.e., we have $\boldsymbol{x} = (x_1, x_2, x_3)$. The Euclidean norm on this space is denoted by $\|\boldsymbol{x}\| = \left(x_1^2 + x_2^2 + x_3^2\right)^{1/2}$. The Euclidean scalar product between two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^3$ is $\boldsymbol{xy} = x_1y_1 + x_2y_2 + x_3y_3$. Furthermore, we denote the componentwise inverse for vectors with non-zero components by $\boldsymbol{x}^{-\mathbf{1}} = \left(x_1^{-1}, x_2^{-1}, x_3^{-1}\right)$ and the componentwise multiplication by $\boldsymbol{x} \odot \boldsymbol{y} = (x_1y_1, x_2y_2, x_3y_3)$. We write $\mathbf{0} = (0, 0, 0)$ and $\mathbf{1} = (1, 1, 1)$.

For a multi-index $\boldsymbol{N} \in 2\mathbb{N}^3$, we set

$$I_{\boldsymbol{N}} = \left\{ \boldsymbol{k} \in \mathbb{Z}^3 \middle| -N_1/2 \leq k_1 < N_1/2, -N_2/2 \leq k_2 < N_2/2, -N_3/2 \leq k_3 < N_3/2 \right\},$$

the index set of possible frequencies.

3 The Coulomb interaction problem

3.1 Task definition

We consider a possibly infinite number of particles, each of which is located at position $\boldsymbol{x}_\ell \in \mathbb{R}^3$ provided with a charge $q_\ell \in \mathbb{R}$. We assume the positions to be all distinct. To store such systems in a computer's memory and to give a meaning to several of its physical quantities, we have to describe it with a finite number of parameters.

We define a *simulation box* to be the parallelepiped spanned by three linearly independent vectors $\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_3 \in \mathbb{R}^3$, namely

$$\{\lambda_1\boldsymbol{b}_1 + \lambda_2\boldsymbol{b}_2 + \lambda_3\boldsymbol{b}_3 \,|\, 0 \leq \lambda_1, \lambda_2, \lambda_3 < 1\}.$$

Within this simulation box, we are given M particles at positions $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M$ and with charges $q_1, \ldots, q_M$. For each box vector, we add the property of *periodicity*, which can be *open* or *periodic*. In the periodic case of box vector $\boldsymbol{b}_\xi$, we replicate the box along this direction, i.e., we consider particles $\boldsymbol{x}_\ell + \mathbb{Z}\boldsymbol{b}_\xi$, all of which have charge q_ℓ, in case of periodicity along box vector $\boldsymbol{b}_\xi$ ($\xi \in \{1, 2, 3\}$). If we do not deal with purely open systems, we require the system to be neutrally charged, i.e., $\sum_{\ell=1}^{M} q_\ell = 0$.

The electrostatic *potential* ϕ is defined as

$$\phi(\boldsymbol{y}) = \sum_{\boldsymbol{x}_\ell \neq \boldsymbol{y}} \frac{q_\ell}{\|\boldsymbol{y} - \boldsymbol{x}_\ell\|} = \sum_\ell q_\ell \mathcal{K}(\boldsymbol{y} - \boldsymbol{x}_\ell), \qquad \text{where } \mathcal{K}(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \boldsymbol{x} = 0, \\ 1/\|\boldsymbol{x}\| & \text{otherwise.} \end{cases} \tag{1}$$

The electrostatic *field* $\boldsymbol{E}$ is the negative gradient of the potential, namely

$$\boldsymbol{E}(\boldsymbol{x}_j) = -\sum_{\boldsymbol{x}_\ell \neq \boldsymbol{y}} q_\ell \frac{\boldsymbol{y}-\boldsymbol{x}_\ell}{\|\boldsymbol{y}-\boldsymbol{x}_\ell\|^3} = -\sum_{\ell} q_\ell \nabla \mathcal{K}(\boldsymbol{y}-\boldsymbol{x}_\ell), \text{ where } \nabla \mathcal{K}(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \boldsymbol{x}=0, \\ \boldsymbol{x}/\|\boldsymbol{x}\|^3 & \text{otherwise.} \end{cases} \tag{2}$$

The electrostatic *force* $\boldsymbol{F}$ is given by $\boldsymbol{F}(\boldsymbol{x}_j) = q_j \boldsymbol{E}(\boldsymbol{x}_j)$.

For this work, we will focus on cubic simulation boxes with box vectors parallel to the principle axes, i.e., we have a box size $B > 0$ and vectors $\boldsymbol{b}_1 = (B,0,0)$, $\boldsymbol{b}_2 = (0,B,0)$ and $\boldsymbol{b}_3 = (0,0,B)$. Purely open systems have already been tested in the serial case [11].

3.2 Convergence matters

The reader will notice that the given definitions are not sufficient to determine the potential and field of a periodic system precisely, since the series in (1) and (2) are only conditionally convergent in general, i.e., the order of summation matters. In fact, for a simple NaCl grid, the sum (1) even diverges if it is summed up along increasing Euclidean norms [2].

The correct summation order according to physical reality is given by ascending cubes [1]. Another approach to avoid this problem would be a three-dimensional Poisson equation with periodic boundary conditions and inhomogeneity given by a sum of translated Dirac distributions [6, Chapter 7.1.2].

Beyond the question of the summation order, the convergence of the series given in (1) and (2) are extremely slow and not useful for numerical calculations.

4 Ewald summation

4.1 Classical Ewald summation

The approach given here was first published by Ewald in 1921 [3]. Its main idea gets clear if we have a closer look at the kernel function mentioned in (1), which has a singularity at the origin and a long-range part causing the slow, conditional convergence. The approach is to split this kernel up into a rapidly decaying short-range part and a smooth long-range part without a singularity. The former term

will converge rapidly in space domain, whereas the latter will do the same in frequency domain. The exact decomposition is $\phi = \phi_1 + \phi_2 - \phi_3$ with

$$\phi_1(\boldsymbol{y}) = \sum_{\boldsymbol{r}\in\mathbb{Z}^3} \sum_{\substack{\ell=1 \\ \boldsymbol{y}\neq\boldsymbol{x}_\ell \text{ for } \boldsymbol{r}=\boldsymbol{0}}}^{M} q_\ell \frac{\operatorname{erfc}(\alpha\|\boldsymbol{y}-\boldsymbol{x}_\ell+\boldsymbol{r}B\|)}{\|\boldsymbol{y}-\boldsymbol{x}_\ell+\boldsymbol{r}B\|},$$

$$\phi_2(\boldsymbol{y}) = \frac{1}{\pi B} \sum_{\boldsymbol{k}\in\mathbb{Z}^3\setminus\{\boldsymbol{0}\}} \frac{\mathrm{e}^{-\pi^2\|\boldsymbol{k}\|^2/(\alpha B)^2}}{\|\boldsymbol{k}\|^2} \sum_{\ell=1}^{M} q_\ell \mathrm{e}^{-2\pi\mathrm{i}\boldsymbol{k}(\boldsymbol{y}-\boldsymbol{x}_\ell)},$$

$$\phi_3(\boldsymbol{y}) = \begin{cases} 2q_\ell\alpha/\sqrt{\pi} & \text{if } \boldsymbol{y}=\boldsymbol{x}_\ell, \\ 0 & \text{if } \boldsymbol{y}\neq\boldsymbol{x}_\ell, \ell=1,\ldots,M \end{cases}$$

and a free splitting parameter $\alpha > 0$. The complementary error function is defined by $\operatorname{erfc}(z) = 2/\sqrt{\pi}\int_z^\infty \mathrm{e}^{-t^2}\mathrm{d}t$. It is now obvious that the convergence of these sums is rapid, so we can cut off some large terms and approximate

$$\phi_1(\boldsymbol{y}) \approx \sum_{\substack{\boldsymbol{r}\in\mathbb{Z}^3,\ell=1,\ldots,M \\ 0<\|\boldsymbol{y}-\boldsymbol{x}_\ell+\boldsymbol{r}B\|<\varepsilon_I}} q_\ell \frac{\operatorname{erfc}(\alpha\|\boldsymbol{y}-\boldsymbol{x}_\ell+\boldsymbol{r}B\|)}{\|\boldsymbol{y}-\boldsymbol{x}_\ell+\boldsymbol{r}B\|}, \tag{3}$$

$$\phi_2(\boldsymbol{y}) \approx \frac{1}{\pi B} \sum_{\boldsymbol{k}\in I_{\boldsymbol{N}}\setminus\{\boldsymbol{0}\}} \frac{\mathrm{e}^{-\pi^2\|\boldsymbol{k}\|^2/(\alpha B)^2}}{\|\boldsymbol{k}\|^2} \sum_{\ell=1}^{M} q_\ell \mathrm{e}^{-2\pi\mathrm{i}\boldsymbol{k}(\boldsymbol{y}-\boldsymbol{x}_\ell)}, \tag{4}$$

where $\varepsilon_I > 0$ is a cutoff-radius in the space domain and $I_{\boldsymbol{N}}$ gives a grid of possible frequencies.

It can now be shown that for a given approximation error and optimal choice of the free parameters α, ε_I and $\boldsymbol{N}$, a direct evaluation of the potentials and fields at all particles via Ewald summation would have a complexity of $\mathcal{O}\left(M^{3/2}\right)$ [10]. There are error estimates for both space and frequency domain [9]. Note that equation (4) differs from [11, Equation (4.9)] because of typing errors.

4.2 An acceleration based on the NFFT

The idea to combine Ewald summation and NFFT was first published by Hedman and Laaksonen in 2006 [7]. The task of a NFFT is to compute the sums

$$f_j = \sum_{\boldsymbol{k}\in I_{\boldsymbol{N}}} \hat{f}_{\boldsymbol{k}} \mathrm{e}^{-2\pi\mathrm{i}\boldsymbol{k}\boldsymbol{x}_j}, \qquad j=1,\ldots,M \tag{5}$$

for a given multi-index $\boldsymbol{N} \in 2\mathbb{N}^3$, at given nodes $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_M \in [0,1)^3$ with given Fourier coefficients $\hat{f}_{\boldsymbol{k}} \in \mathbb{C}$ for $\boldsymbol{k} \in I_{\boldsymbol{N}}$. The corresponding adjoint (in the sense of transposed and complex conjugated) task is to evaluate the sums

$$\hat{f}_{\boldsymbol{k}} = \sum_{j=1}^{M} f_j \mathrm{e}^{2\pi\mathrm{i}\boldsymbol{k}\boldsymbol{x}_j}, \qquad \boldsymbol{k}\in I_{\boldsymbol{N}} \tag{6}$$

for a given multi-index $\boldsymbol{N} \in 2\mathbb{N}^3$ given nodes $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M \in [0,1)^3$ and given values $f_j \in \mathbb{C}$ for $j = 1, \ldots, M$. If we have algorithms which perform both NFFT and adjoint NFFT, the evaluation of (4) reduces to the following algorithm.

Algorithm 1 (Evaluation of the Ewald far-field part)**.** Given parameters are the Ewald splitting parameter $\alpha > 0$ and the possible frequencies determined by $\boldsymbol{N} \in \mathbb{N}^3$.

1. Set the nodes $\boldsymbol{x}_1/B, \ldots, \boldsymbol{x}_M/B \in [0,1)^3$.
2. Set the values $f_j \leftarrow q_j$ for $j = 1, \ldots, M$.
3. Let $S(\boldsymbol{k}) = \sum_{j=1}^{M} f_j \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}_j}$, $k \in I_{\boldsymbol{N}}$, be the result of an adjoint NFFT with the given nodes and values.
4. Set $S(\boldsymbol{0}) \leftarrow 0$ and multiply each $S(\boldsymbol{k})$ with the coefficients
$$\hat{b}_{\boldsymbol{k}} = \frac{\mathrm{e}^{-\pi^2 \|\boldsymbol{k}\|^2/(\alpha B)^2}}{\pi B \|\boldsymbol{k}\|^2} \qquad (\boldsymbol{k} \in I_{\boldsymbol{N}} \setminus \{\boldsymbol{0}\}).$$
5. The required approximated potentials $\phi(\boldsymbol{x}_j)$ are given as the result of an NFFT applied to the modified Fourier coefficients $S(\boldsymbol{k})$ at the given nodes $\boldsymbol{x}_j$, $j = 1, \ldots, M$.

4.3 The basics of the NFFT algorithm

We follow the procedure described in [13, Section 1.1] and [11, Section 2] and start with a well-localised continuous window function $\varphi \in \mathrm{L}^2(\mathbb{R}^3) \cap \mathrm{L}^1(\mathbb{R}^3)$, for example a multivariate Gaussian, such that its periodic version
$$\tilde{\varphi}(\boldsymbol{x}) = \sum_{\boldsymbol{r} \in \mathbb{Z}^3} \varphi(\boldsymbol{x} + \boldsymbol{r})$$
has a uniformly convergent Fourier series
$$\tilde{\varphi}(\boldsymbol{x}) = \sum_{\boldsymbol{k} \in \mathbb{Z}^3} c_{\boldsymbol{k}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}} \qquad \text{with} \qquad c_{\boldsymbol{k}} = \int_{[0,1)^3} \tilde{\varphi}(\boldsymbol{x}) \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}} \mathrm{d}\boldsymbol{x}, \qquad \boldsymbol{k} \in \mathbb{Z}^3.$$
Let $\boldsymbol{\sigma} \in \mathbb{R}^3$ be an oversampling factor with $\sigma_1, \sigma_2, \sigma_3 > 1$, such that $\boldsymbol{n} = \boldsymbol{\sigma} \odot \boldsymbol{N} \in \mathbb{N}^3$. We wish to approximate the sum (5) by the linear combination
$$s_1(\boldsymbol{x}) = \sum_{\boldsymbol{\ell} \in I_{\boldsymbol{n}}} g_{\boldsymbol{\ell}} \tilde{\varphi}\left(\boldsymbol{x} - \boldsymbol{n}^{-\boldsymbol{1}} \odot \boldsymbol{\ell}\right)$$
with $g_{\boldsymbol{\ell}} \in \mathbb{C}$ ($\boldsymbol{\ell} \in I_{\boldsymbol{n}}$). By the Fourier expansion of $\tilde{\varphi}$, we obtain
$$\begin{aligned} s_1(\boldsymbol{x}) &= \sum_{\boldsymbol{k} \in \mathbb{Z}^3} \sum_{\boldsymbol{\ell} \in I_{\boldsymbol{n}}} g_{\boldsymbol{\ell}} c_{\boldsymbol{k}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}\left(\boldsymbol{x} - \boldsymbol{n}^{-\boldsymbol{1}} \odot \boldsymbol{\ell}\right)} = \sum_{\boldsymbol{k} \in \mathbb{Z}^3} \hat{g}_{\boldsymbol{k}} c_{\boldsymbol{k}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}} \\ &= \sum_{\boldsymbol{k} \in I_{\boldsymbol{n}}} \hat{g}_{\boldsymbol{k}} c_{\boldsymbol{k}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}} + \sum_{\boldsymbol{r} \in \mathbb{Z}^d \setminus \{\boldsymbol{0}\}} \sum_{\boldsymbol{k} \in I_{\boldsymbol{n}}} \hat{g}_{\boldsymbol{k}} c_{\boldsymbol{k} + \boldsymbol{n} \odot \boldsymbol{r}} \mathrm{e}^{-2\pi \mathrm{i} (\boldsymbol{k} + \boldsymbol{n} \odot \boldsymbol{r}) \boldsymbol{x}} \end{aligned}$$

with

$$\hat{g}_{\boldsymbol{k}} = \sum_{\boldsymbol{\ell} \in I_{\boldsymbol{n}}} g_{\boldsymbol{\ell}} \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{k} \left(\boldsymbol{n}^{-\mathbf{1}} \odot \boldsymbol{\ell} \right)}, \qquad \boldsymbol{k} \in I_{\boldsymbol{n}}.$$

Comparing the result to (5), we would like to choose

$$\hat{g}_{\boldsymbol{k}} = \begin{cases} \hat{f}_{\boldsymbol{k}} / c_{\boldsymbol{k}} & \text{if } \boldsymbol{k} \in I_{\boldsymbol{N}}, \\ 0 & \text{otherwise} \end{cases}$$

if the Fourier coefficients $c_{\boldsymbol{k}}$ decay rapidly. The inversion formula of the discrete Fourier transform shows that then we have

$$g_{\boldsymbol{\ell}} = \frac{1}{n_1 n_2 n_3} \sum_{\boldsymbol{k} \in I_N} \hat{g}_{\boldsymbol{k}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \left(\boldsymbol{n}^{-\mathbf{1}} \odot \boldsymbol{\ell} \right)}, \qquad \boldsymbol{\ell} \in I_{\boldsymbol{n}},$$

which can be computed by a usual three-dimensional FFT.

After the cutoff in Fourier space, we truncate our window function φ in the real space by defining

$$\psi(\boldsymbol{x}) = \begin{cases} \varphi(\boldsymbol{x}) & \text{if } -m_\xi / n_\xi \leq x_\xi < m_\xi / n_\xi, \xi = 1,2,3, \\ 0 & \text{otherwise} \end{cases}$$

with a truncation parameter $\boldsymbol{m} \in \mathbb{N}^3$ subject to $m_\xi \ll n_\xi$ for $\xi = 1,2,3$ and forming a periodic version

$$\tilde{\psi}(\boldsymbol{x}) = \sum_{\boldsymbol{r} \in \mathbb{Z}^3} \psi(\boldsymbol{x} + \boldsymbol{r}).$$

Now we have

$$s_1(\boldsymbol{x}) = \sum_{\boldsymbol{\ell} \in I_{\boldsymbol{n}}} g_{\boldsymbol{\ell}} \tilde{\varphi}\left(\boldsymbol{x} - \boldsymbol{n}^{-\mathbf{1}} \odot \boldsymbol{\ell} \right) \approx \sum_{\boldsymbol{\ell} \in I_{\boldsymbol{n}}} g_{\boldsymbol{\ell}} \tilde{\psi}\left(\boldsymbol{x} - \boldsymbol{n}^{-\mathbf{1}} \odot \boldsymbol{\ell} \right) = \sum_{\boldsymbol{\ell} \in I_{\boldsymbol{n},\boldsymbol{m}}(\boldsymbol{x})} g_{\boldsymbol{\ell}} \tilde{\psi}\left(\boldsymbol{x} - \boldsymbol{n}^{-\mathbf{1}} \odot \boldsymbol{\ell} \right)$$

with the index set

$$I_{\boldsymbol{n},\boldsymbol{m}}(\boldsymbol{x}) = \{ \boldsymbol{\ell} \in I_{\boldsymbol{n}} \mid n_\xi x_\xi - m_\xi \leq \ell_\xi \leq n_\xi x_\xi + m_\xi, \xi = 1,2,3 \}$$

as a final approximation for the sum (5).

Algorithm 2 (NFFT)**.** Given parameters are M nodes $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M$, the multi-index $\boldsymbol{N} \in 2\mathbb{N}^3$ defining the possible frequencies, an oversampling factor $\boldsymbol{\sigma}$, a truncation parameter $\boldsymbol{m} \in \mathbb{N}^3$ and the Fourier coefficients $\hat{f}_{\boldsymbol{k}} \in \mathbb{C}$ for $\boldsymbol{k} \in I_{\boldsymbol{N}}$. We define $\boldsymbol{n} = \boldsymbol{\sigma} \odot \boldsymbol{N}$.

1. Compute the Fourier coefficients $c_{\boldsymbol{k}}$, $\boldsymbol{k} \in I_{\boldsymbol{N}}$ of the window function.
2. Compute the values $\tilde{\psi}\left(\boldsymbol{x}_j - \boldsymbol{n}^{-\mathbf{1}} \odot \boldsymbol{\ell} \right)$ for $j = 1, \ldots, M$ and $\boldsymbol{\ell} \in I_{\boldsymbol{n},\boldsymbol{m}}(\boldsymbol{x}_j)$
3. Let $\hat{g}_{\boldsymbol{k}} \leftarrow \hat{f}_{\boldsymbol{k}} / c_{\boldsymbol{k}}$, $\boldsymbol{k} \in I_{\boldsymbol{N}}$ and $\hat{g}_{\boldsymbol{k}} \leftarrow 0$ for $\boldsymbol{\ell} \in I_{\boldsymbol{n}} \setminus I_{\boldsymbol{N}}$.
4. Compute

$$g_{\boldsymbol{\ell}} = \frac{1}{n_1 n_2 n_3} \sum_{\boldsymbol{k} \in I_N} \hat{g}_{\boldsymbol{k}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \left(\boldsymbol{n}^{-\mathbf{1}} \odot \boldsymbol{\ell} \right)}, \qquad \boldsymbol{\ell} \in I_{\boldsymbol{n}}$$

   by an FFT.

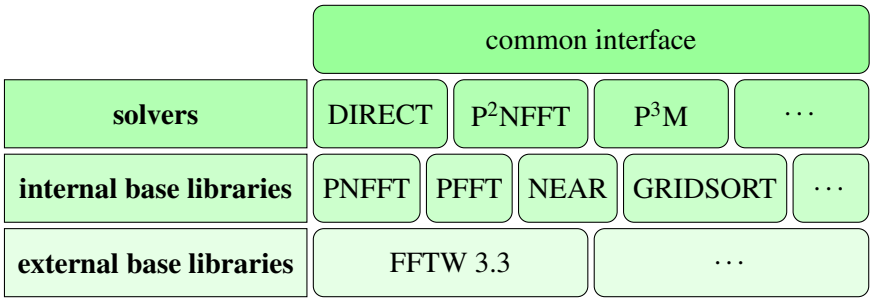


Figure 1: The modular structure of the ScaFaCoS package.

5. The approximative results are obtained by

$$f_j \approx \sum_{\boldsymbol{\ell} \in I_{\boldsymbol{n},\boldsymbol{m}}(\boldsymbol{x})} g_{\boldsymbol{\ell}} \tilde{\psi} \left(\boldsymbol{x} - \boldsymbol{n}^{-\mathbf{1}} \odot \boldsymbol{\ell} \right).$$

Step 1 can be done as a pre-computation and is needed only once. If the nodes remain unchanged, the same is true for the calculation of $\tilde{\psi}$. It is easily seen that the complexity of this algorithm is $\mathcal{O}\left(n_1 n_2 n_3 \log\left(n_1 n_2 n_3\right) + m_1 m_2 m_3 M\right)$ and thus faster than the direct calculation with a complexity of $\mathcal{O}\left(M N_1 N_2 N_3\right)$ if $\boldsymbol{\sigma}$ and $\boldsymbol{m}$ have a bounded size.

For further details, such as an algorithm for the adjoint NFFT, different window functions φ and error estimates, see for example [13, Chapter 1]. In order to get the electrostatic fields by Ewald summation, one can efficiently calculate gradients of the trigonometric polynomial given in (5) by deriving the window function, see [8, Chapter 8-3-2].

5 Implementation details

5.1 On the ScaFaCoS library

The ScaFaCoS library [15] is designed as an open-source collection of state-of-the art algorithms for the Coulomb interaction problem with different box shapes and periodicity conditions. To achieve maximal usability, all the solvers can be called by a common interface.

The modular structure shown in Figure 1 ensures that the subset of actually needed solvers can be compiled if all required external components are available.

The solver mentioned in the title of this report is P^{2}NFFT. It depends on the PNFFT and PFFT auxiliary libraries [12] developed by Michael Pippig and the near-field solver by Olaf Lenz, René Halver and Michael Hofmann. The PNFFT and PFFT themselves depend on the well-known FFTW subroutine library [4] in version 3.3 with MPI support.

Further solvers contained in the ScaFaCoS package include a direct solver, which is suitable for small non-periodic systems and for testing purposes, and the P^{3}M solver, which is also based on an Ewald summation. We will use both of them to estimate the accuracy of P^{2}NFFT.

5.2 The P²NFFT solver

Our implementation computes the potentials and forces (1) and (2) of a system described at the end of Section 3.1 by the Ewald splitting mentioned in Section 4.1. The sum (4) is calculated by Algorithm 1 where the NFFT and adjoint NFFT are performed by the PNFFT library, which provides a parallel implementation of Algorithm 2. For an efficient calculation of the near-field sum (3), the ScaFaCoS near-field library is used.

The index set of the NFFT implemented in PNFFT is different from that we stated in Section 2, namely

$$I'_{\boldsymbol{N}} = \left\{ \boldsymbol{k} \in \mathbb{Z}^3 \middle| \, 0 \le k_1 < N_1, 0 \le k_2 < N_2, 0 \le k_3 < N_3 \right\}$$

is used. Thus, P²NFFT utilises the bijection $I'_{\boldsymbol{N}} \to I_{\boldsymbol{N}}, \boldsymbol{k'} \mapsto \boldsymbol{k}$ where

$$k_\xi = \begin{cases} k'_\xi & \text{if } 0 \le k'_\xi < N_\xi/2, \\ k'_\xi - N_\xi & \text{otherwise,} \end{cases} \qquad \xi = 1, 2, 3.$$

After that, step 4 of Algorithm 1 can directly be applied to the transformed indices.

Since we want to compare the results of the P²NFFT and P³M solvers, we chose the oversampling factor to be $\boldsymbol{\sigma} = \mathbf{1}$, which is possible for a Gaussian window function φ currently used. The error estimates in [9] show that this choice of parameters still gives a reasonable result. The tuning routine, which selects values for α, ε_I, $\boldsymbol{N}$ and $\boldsymbol{m}$ according to a desired truncation error, was adopted from the P³M solver and first developed in the ESPResSo project [14].

To complete the parallel P²NFFT solver, the nodes $\boldsymbol{x}_j$ need to be sorted to fit the parallel data distribution of the FFT grid. Since we were not able to finish the integration of the ScaFaCoS parallel sorting library during the guest student programme, our test programs simply exclude all the particles not needed on the respective process.

6 Numerical results

6.1 NaCl grids

We now consider a simple grid structure consisting of a cubic box of size $B \in 2\mathbb{N}$. For simplicity of notation, we index our particles by $\boldsymbol{j} \in \{0, \dots, B-1\}^3$ and set the positions and charges as

$$\boldsymbol{x_j} = (0.5, 0.5, 0.5) + \boldsymbol{j} \qquad \text{and} \qquad q_{\boldsymbol{j}} = \begin{cases} +1 & \text{if } j_1 + j_2 + j_3 \text{ is even,} \\ -1 & \text{otherwise.} \end{cases}$$

For these systems, the theoretical results are well-known: All the forces are equal to $\mathbf{0}$ due to symmetry reasons, and the potentials at the positions of the particles are given by $\phi(\boldsymbol{x_j}) = M q_{\boldsymbol{j}}$, where $M = 1.747564594633\dots$ is the Madelung constant of the grid. The results for different grid sizes and desired accuracies are given in Tables 1 and 2.

Accuracy	α	ε_I	m	N	ϕ	error $\lvert\phi - M\rvert / M$
1	0.416277	2.0	1	4	1.0736912297	$3.856 \cdot 10^{-1}$
$1 \cdot 10^{-1}$	0.865409	2.0	2	4	1.7079629581	$2.266 \cdot 10^{-2}$
$1 \cdot 10^{-2}$	1.150904	2.0	5	4	1.7452950265	$1.299 \cdot 10^{-3}$
$1 \cdot 10^{-3}$	1.378487	2.0	4	8	1.7474337868	$7.485 \cdot 10^{-5}$
$1 \cdot 10^{-4}$	1.573490	2.0	7	8	1.7475584441	$3.519 \cdot 10^{-6}$
$1 \cdot 10^{-5}$	1.746860	2.0	4	32	1.7475843710	$1.132 \cdot 10^{-5}$
$1 \cdot 10^{-6}$	1.904512	2.0	5	32	1.7475686878	$2.342 \cdot 10^{-6}$
$1 \cdot 10^{-7}$	2.050076	2.0	4	128	1.7476323042	$3.875 \cdot 10^{-5}$

Table 1: Tuning results and errors for serial calculation of an NaCl grid, $B = 2$

Accuracy	α	ε_I	m	N	ϕ	error $\lvert\phi - M\rvert / M$
1	0.233432	3.0	1	4	2.9413864713	$6.831 \cdot 10^{-1}$
$1 \cdot 10^{-1}$	0.557076	3.0	2	16	1.7983182215	$2.904 \cdot 10^{-2}$
$1 \cdot 10^{-2}$	0.752447	3.0	3	32	1.7508668536	$1.890 \cdot 10^{-3}$
$1 \cdot 10^{-3}$	0.906653	3.0	5	32	1.7478080096	$1.393 \cdot 10^{-4}$
$1 \cdot 10^{-4}$	1.038201	3.0	4	128	1.7475829304	$1.049 \cdot 10^{-5}$
$1 \cdot 10^{-5}$	1.154861	3.0	6	64	1.7475660880	$8.545 \cdot 10^{-7}$
$1 \cdot 10^{-6}$	1.260772	3.0	6	128	1.7475645343	$3.452 \cdot 10^{-8}$

Table 2: Tuning results and errors for serial calculation of an NaCl grid, $B = 16$

6.2 Irregular structures

To test the implementation for forces and the parallelisation, we looked into systems with a less regular distribution of the particles. We will not compare our results to exact results but to those of the P^{3}M solver, for which the considered system was a test case. It contains $M = 300$ particles and is given in Figure 2. We measured numerical values for the total energy

$$E = \frac{1}{2} \sum_{\ell=1}^{M} q_\ell q_\ell \phi\left(\boldsymbol{x}_\ell\right)$$

and the root-mean-square deviation

$$d = \sqrt{\frac{1}{M} \sum_{\ell=1}^{M} \left\| \boldsymbol{F}_\ell - \widetilde{\boldsymbol{F}}_{\boldsymbol{\ell}} \right\|^2}$$

of the calculated forces $\widetilde{\boldsymbol{F}}_\ell$ to some given reference forces $\boldsymbol{F}_\ell$ ($\ell = 1, \ldots, M$), see Table 3. Up to the limited precision of the forces and apparent accuracy restrictions of P^{3}M, the values for both solvers coincide. These results still hold if they are computed by up to 32 parallel processes.

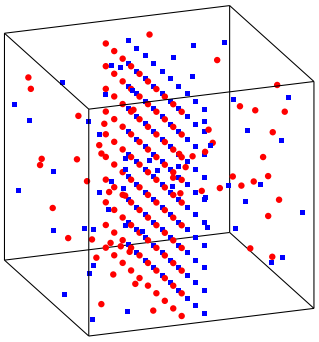


Figure 2: Simulation box of the test system for Section 6.2. Red dots and blue squares represent positive and negative unit charges, respectively.

	P^{2}NFFT		P^{3}M	
Accuracy	E	d	E	d
1	314.770544	$3.203 \cdot 10^{-1}$	314.770544	$3.203 \cdot 10^{-1}$
$1 \cdot 10^{-1}$	151.804315	$1.098 \cdot 10^{-1}$	164.799587	$1.129 \cdot 10^{-1}$
$1 \cdot 10^{-2}$	149.267852	$8.580 \cdot 10^{-3}$	150.654065	$1.254 \cdot 10^{-2}$
$1 \cdot 10^{-3}$	148.966722	$9.319 \cdot 10^{-4}$	148.892301	$9.257 \cdot 10^{-4}$
$1 \cdot 10^{-4}$	148.945429	$8.529 \cdot 10^{-5}$	148.937486	$7.425 \cdot 10^{-5}$
$1 \cdot 10^{-5}$	148.936785	$8.823 \cdot 10^{-4}$	148.942824	$4.383 \cdot 10^{-5}$
$1 \cdot 10^{-6}$	148.941848	$2.117 \cdot 10^{-4}$	126.203799	$6.114 \cdot 10^{-1}$
$1 \cdot 10^{-7}$	148.940754	$3.637 \cdot 10^{-4}$	124.327306	$6.379 \cdot 10^{-1}$

Table 3: Total energies and root-mean-square deviations of the forces for the test system

7 Conclusions and outlook

We have implemented a solver which computes electrostatic potentials and fields for cubic periodic systems of charged particles. The solver works together with the ScaFaCoS interface and gives correct results for all implemented test cases. In contrast to the P^3M solver, the implemented P^2NFFT solver can handle better accuracies. Time and scaling measurements were not sensible in the current state of development of P^2NFFT and not comparable to other ScaFaCoS solvers.

We will give a short overview of further desired functionality concerning the new solver.

- An efficient parallelisation needs integration of the gridsort auxiliary library into P^2NFFT.
- In order to optimise the runtime, a pre-computation step could choose an optimal balance between the times needed for the real space part (3) and the Fourier space part (4). This would be done by measuring some computation times similar to the flag `FFTW_MEASURE` of the FFTW [5, Chapter 4.3.2].
- Purely open or mixed-periodic systems could be handled by the formulae given in [11, Chapter 4.1] or as tensor products of both approaches, respectively, but it should be investigated, whether we have to adapt the error estimates.
- It could be checked if the Ewald splitting can be generalised to other configurations of box vectors, e.g. non-orthogonal crystals.

8 Acknowledgements

The author would like to thank René Halver, Michael Pippig and Godehard Sutmann for their supervision during the guest student programme and many helpful remarks as well as Mathias Winkel and Natalie Schröder for the excellent organisation and Daniel Potts for recommending him for the programme.

The development of ScaFaCoS is supported by BMBF grant 01IH08001B.

References

1. D. BORWEIN, J. M. BORWEIN and K. F. TAYLOR, *Convergence of lattice sums and Madelung's constant*, J. Chem. Phys. **26**, 2999–3009 (1985).
2. O. EMERSLEBEN, *Über die Konvergenz der Reihen Epsteinscher Zetafunktionen*, Math. Nachr. **4**, 468–480 (1951).
3. P. P. EWALD, *Die Berechnung optischer und elektrostatischer Gitterpotentiale*, Ann. Phys. **64**, 253–287 (1921).
4. M. FRIGO and S. G. JOHNSON, *FFTW, C subroutine library*, `http://www.fftw.org`.
5. M. FRIGO and S. G. JOHNSON, *FFTW*, user manual for version 3.3, `http://www.fftw.org/fftw3.pdf` (2011).
6. M. GRIEBEL, S. KNAPEK and G. ZUMBUSCH, *Numerical simulation in molecular dynamics*, vol. 5 of *Texts in Computational Science and Engineering*, Springer, Berlin (2007).
7. F. HEDMAN and A. LAAKSONEN, *Ewald summation based on nonuniform fast Fourier transform*, Chem. Phys. Lett. **425**, 142–147 (2006).
8. R. W. HOCKNEY and J. W. EASTWOOD, *Computer simulation using particles*, Taylor & Francis Inc., Bristol, PA, USA (1988).

9. J. KOLAFA, J. W. PERRAM, *Cutoff Errors in the Ewald Summation Formulae for Point Charge Systems*, Molecular Simulation **9**, 351–368 (1992).
10. J. W. PERRAM, H. PETERSEN and S. DE LEEUW, *An algorithm for the simulation of condensed matter which grows as the 3/2 power of the number of particles*, Mol. Phys. **65**, 875–893 (1988).
11. M. PIPPIG and D. POTTS, *Particle Simulation Based on Nonequispaced Fast Fourier Transforms*, in G. SUTMANN, P. GIBBON, T. LIPPERT (eds.), *Fast Methods for Long-Range Interactions in Complex Systems*, 131–158 (2011).
12. M. PIPPIG, *Software,* `http://www.tu-chemnitz.de/~mpip/software.php`.
13. D. POTTS, *Schnelle Fourier-Transformation für nichtäquidistante Daten und Anwendungen*, Habilitation, Universität zu Lübeck (2003).
14. *ESPResSo Extensible Simulation Package for Research on Soft matter*, `http://espressomd.org/wiki/Main_Page`.
15. *ScaFaCoS Scalable Fast Coulomb Solver*, `http://www2.fz-juelich.de/jsc/scafacos`.

Viscous flow through fractal contacts

Andreas Lücke

Universität Paderborn
Warburger Str. 100
33098 Paderborn

E-mail: ALuecke@gmx.net

Abstract:
In this report we investigate and simulate the fluid leakage through a seal. The flow is obtained by numerically solving the Reynold's lubrication equation for fractal contacts. Fractal topograhies are generated with different Hurst-roughness exponents in order to create simulation cells. The influence of an external force pressing a flat elastic body against rigid substrates is calculated with the Green's function molecular dynamics (GFMD) and the overlap-model. Finally the flow currents for the two models are compared.

---

1 Introduction

Seals are used e.g. to connect pipes. In this case the gap between the pipes needs to be closed or sealed in the best possible way. Malfunction of a seal can have dramatic consequences, as we have witnessed for the decline of the oil rig "Deep Water Horizon". Overall the development process of seals bases foremost on experimental investigations which is expensive and time-consuming. Therefore we want to use numerical techniques to study the performance of seals. Unfortunately a surface is rough so when you zoom in you see contact only at a few points. Therefore the most important problem in seals is the influence of the surface roughness to the effective contact area. Because of the fine geometry you want to resolve you need a fine discretisation. That is why many grid points are needed which have to be updated every iteration which effects high computational costs. In this report a parallel MPI-implementation will be presented which is able to solve the problem on highly parallel machines.

2 Theoretical fundamentals

In this work the Reynold's lubrication equation is used in order to calculate the fluid flow through a fractal contact (one plain and elastic surface is pressed on another rough surface):

$$\nabla \cdot \mathbf{J} = \nabla \cdot \left(\frac{u(x,y)^3}{12\eta} \cdot \nabla p(x,y) \right) = 0 \tag{1}$$

Thereby **J** is the current through the seal, p the pressure, $u(x, y)$ the gap height of the seal at the position x, y und η the viscosity. This equation can be derived from the Navier-Stokes equation. You can comprehend the equation as well in a plausible way if you compare the different terms with Ohm's law:

$$I = \sigma \cdot U = \frac{1}{R} \cdot U \tag{2}$$

To let a fluid flow a pressure difference is needed, because at constant pressure exists no force which could move the fluid. In comparison to Ohm's law ∇p is equivalent to the voltage. Furthermore, the fluid flow will increase with a growing gap height, because the fluid has more space to flow. Hence the conductivity in Ohm's law is correlated to the gap height $u(x, y)$ in the Reynold's lubrication equation. The viscosity inversely affects the current because a fluid flows worse the more viscous it is. (E.g. heavy oil will result in a much smaller current through a seal than water.) Because of the continuity equation a current doesn't have a source or a sink. That's why the divergence of **J** has to be zero.
By expanding the Reynold lubrication equation you obtain an equation which is similar to the Poisson-equation:

$$\iff \Delta p(x, y) = -\frac{12\eta}{u(x, y)^3} \cdot \nabla \left(\frac{u(x, y)^3}{12\eta} \right) \tag{3}$$

Hence several solvers for the Poisson-equation can also be used for the Reynold lubrication equation.

3 Experimental setup

In the following subsections further details to the implementation are given.

3.1 Boundary conditions

While numerically solving the Reynold's lubrication equation we assume periodic boundary conditions along y-axis. Furthermore the pressure is set to 1 at $x = 0$ and 0 at $x = L$. Due to this pressure difference the fluid will flow in the bigger positive x-direction.

The derivatives in the partial differential equation are approximated with differential quotients on the discrete grid.

3.2 Solver for the Reynolds lubrication equation

In this part two different solvers for the partial differential equation will be presented.
Following there will be an error definition given in order to estimate the error or accuracy of a solution:

$$\chi^2_{local} = (\nabla \cdot J_{local})^2 \tag{4}$$

$$\chi^2_{global} = \sum_{i \in GridPoints} (\nabla \cdot J_{local,i})^2 \tag{5}$$

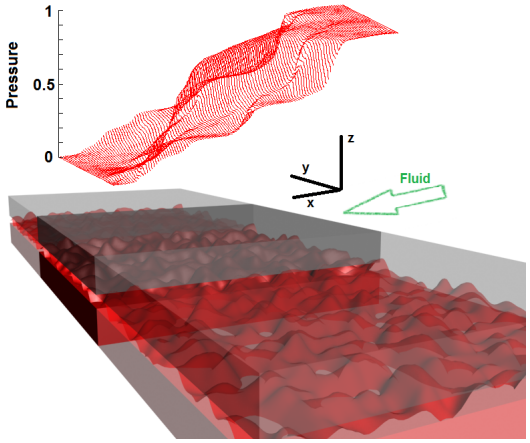


Figure 1: The translucent simulation cells represent the periodic boundary condition along y-axis. The graph on the top depicts the pressure course with $p(0) = 1$ and $p(L) = 0$. The fluid flows from high to low pressure.

Ideally the divergence of the current is zero if the solution has converged. That is why we can interprete the deviant to zero as a local error. The global error is only the sum of all local errors. If this error is smaller than 10^{-12} the solution is contemplated to be converged.

3.2.1 Local solver

As the name already reveals this solver is used to solve the problem only local.

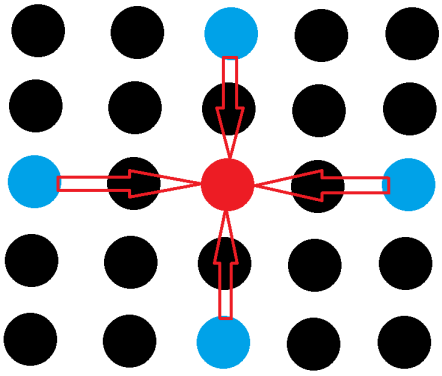


Figure 2: The new pressure for a local point is only calculated under inclusion of the next-nearest neighbours.

Therefore we iterate over all gridpoints and calculate the local error for each point with the help of the next-nearest neighbours. The local error is only linear in the local pressure so the latter can be changed in order to get the local error equals zero. This solver is NOT converged after one iteration over all grid points, because e.g. by changing the local pressure of the next nearest neighbour $(x + 2, y)$ you get again an error on (x, y). So in order to converge you have to do much more iterations over all grid points. The disadvantage of this method is the extreme slow diffusion of "information" through the system. Until a point in the middle of the system recognizes that at $x = 0$ a pressure is equal to 1 you need many iterations, because every point calculates its pressure only respecting the next-nearest

neighbours. Hence the information diffuses only $\sim \mathcal{O}(L^2)$ through the system and each iteration costs $\mathcal{O}(L^2)$ the total effort is $\mathcal{O}(L^4)$ which is an unwanted scaling behavior: A doubled system size causes a by factor 16 increased effort.

3.2.2 Fourier solver

In order to bypass the disadvantages of the local solver another solver been implemented which works in fourier space.

1. At first a local "force" is beeing calculated:

$$F_{local} = \frac{\delta \chi^2_{local}}{\delta p} \tag{6}$$

2. After this the force is Fourier transformed with the `fftw3`-library:

$$F_{local}(\mathbf{r}) \longrightarrow \widetilde{F}_{local}(\mathbf{q}) \tag{7}$$

3. In Fourier space we can stress the long wavelengths (small frequencies q) compared to the shorter ones (great frequencies q):

$$\widehat{\widetilde{F}}_{local}(\mathbf{q}) = \frac{\widetilde{F}_{local}(\mathbf{q})}{(\frac{2\pi}{L} + q)^{\alpha}} \tag{8}$$

   α is a free parameter which can be accommodated in order to get better convergence. In the equation are modes with smaller wavelengths understated by dividing them with a greater number than modes with a greater wavelength (smaller q). By using this transformation the slow diffusion of information should be avoided, because the long wavelength-modes should spread the information in the system much faster.

4. Finally the modified force is transformed back and the pressure gets updated. The free parameter t is used to minimize the global error (best deepest decent). Therefore the global error is calculated for the factors t, $-t$ and 0 and then the angular point of the fitted parabola is used.

$$p_{n+1} = p_n + t \cdot FFT(\widehat{\widetilde{F}}_{local}(\mathbf{q})) \tag{9}$$

   Together with the local solver the new hybrid version should converge much faster.

A proper choice of the parameter α can reduce the number of iterations dramatically as depicted in figure 3. For $\alpha = 0.55$ almost 10% of the original number of iterations are needed. These good results are only obtained for easy geometries from the fourier solver. For real fractals the Fourier solver unfortunately looses its speed advantage because in this case there exists no characteristic lengthscale any longer. That is why only the local solver is used for the following calculations.

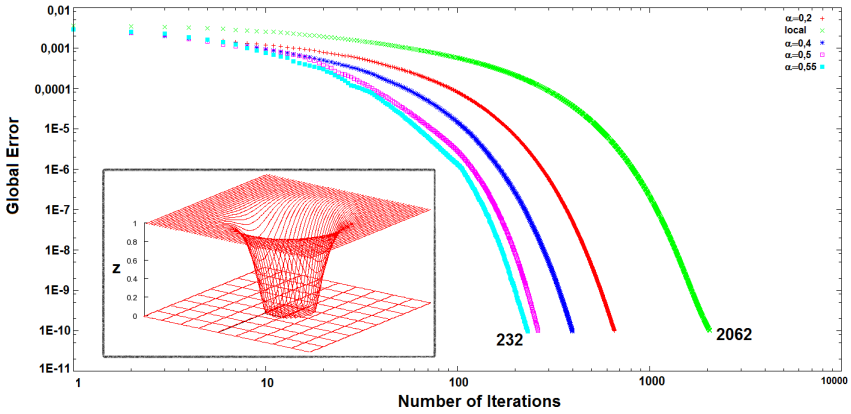


Figure 3: Convergence of the hybrid version for different α's. The green (rightest) graph depicts the convergence of the pure local solver. The calculation was done for the geometry in the left bottom corner.

3.2.3 Parallelizing the local solver with MPI

Beside a small parallelization with OpenMP the local solver has been parallized with MPI in order to make the programm suitable for highly parallel machines. In order to parallelize the local solver the 2D area has to be splitted among the processors. There exist two possibilities:

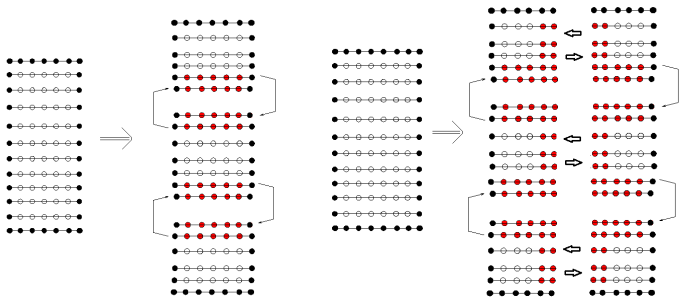


Figure 4: Possible decomposition in stripes and blocks.

The need of the next nearest neighbours in order to update the local pressure values causes a problem in the boundary points of the decomposition hence the neighbour values are stored at other processes. Therefore we introduce ghost points which have to be exchanged between neighboured processes after each iteration.

If you analyze the scaling behavior of the 1D and 2D decomposition you will find out that the latter one scales better with an increasing number of processes (done in [6]). That is why a 2D decomposition is used in this work. In order to obtain a 2D decomposition of the area the method `MPI_Cart_create()` is used which ideally takes care that neighboured points in the grid are also calculated on neighboured processors. This reduces the communication costs. Furthermore you can state periodical boundary conditions along an arbitrary axis. `MPI_Cart_shift()` is used to get neighboured processes which are needed to exchange the ghost points. The method automatically takes care of the stated boundary conditions. The data for the ghost points of the left and the right process are not stored contigously in memory because the pressure array is stored row-wise. By creating

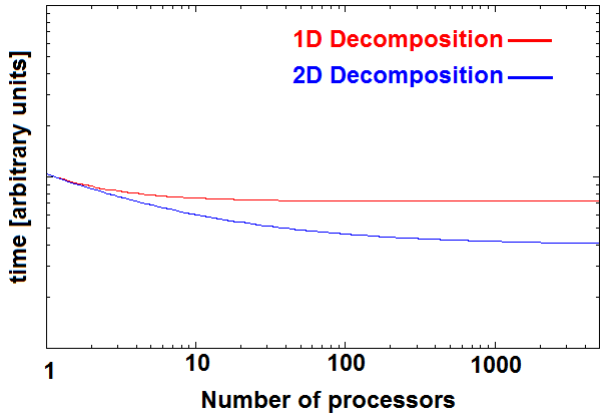


Figure 5: Scaling behavior of the two different decompositions.

an extra datatype with `MPI_Type_vector()` an extra copy of the ghostpoints to a contigous array can be avoided. For storing the solution MPI file I/O is used. Each process obtains a window inside the file in which it can write its data. After this the pressure is stored in the right order in a binary file.

3.3 Generating fractal topographies

In order to generate fractal topographies modes inside a circle $|\mathbf{q}| < q_{cut}$ in 2D Fourier space become weighted by Gaussian numbers. Algorithmus for the heigth map $G(\mathbf{r})$:

1. Get 2 Gaussian numbers g_1, g_2 (for example with Box Mueller-algorithm)
2. calculate:
   - $\mathcal{R}\left(\tilde{G}(\mathbf{q})\right) = g_1 \cdot q^{-(H+1)}$
   - $\mathcal{I}\left(\tilde{G}(\mathbf{q})\right) = g_2 \cdot q^{-(H+1)}$

   H is the Hurst roughness exponent which is a number in the range $[0, 1]$. For a larger Hurst roughness exponent the fractal surface becomes smoother, because the roughness lives more on the long lengthscales. From the equation follows that when using smaller H, the larger q are less suppressed so the surface is much finer (compare figure 6). In reality typical surfaces have a Hurst roughness exponent similar to 0.8.
3. Get $G(\mathbf{r}) = \mathrm{FFT}\left(\tilde{G}(\mathbf{q})\right)$
   Note that $G(\mathbf{q})$ must be hermitian, otherwise $G(\mathbf{r})$ is not real after back-transformation.

3.4 Influence of an external force

The influence of an external force on a surface can be calculated among others with 2 models:

- overlap model

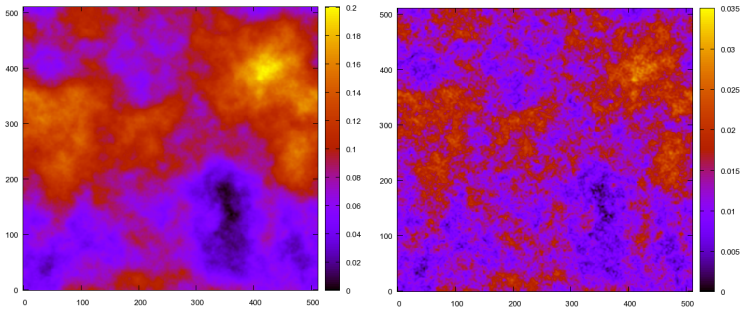


Figure 6: Left: $H = 0.8$ Right: $H = 0.2$

- Greens function molecular dynamic (GFMD)

3.4.1 Overlap model

The external force presses a soft, deformable surface against a undeformable surface. According to the overlap model all (invalid) points which would be inside the other material get cut and their value is set to position of the surface. Therefore a sharp edge (right side figure 7) is generated at the border of the contact. This model is often used by many engineers in order to calculate contact mechanics.

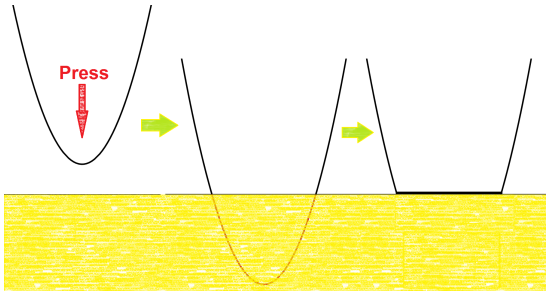


Figure 7: Calculating the influence of an external force according to the overlap model.

3.4.2 GFMD model

In the GFMD model the displacement of a grid point has an influence on all other grid points. Hence you do not get an sharp edge because the border points get “hauled up” by the displaced neighboured points. In the GFMD-theory the full, soft material can be reduced to its surface, because the third dimension can be out integrated for further calculations. This model produces results which pay much more attention to the real physical behavior of materials than the overlap model.

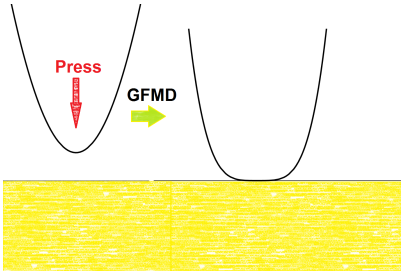


Figure 8: Calculating the influence of an external force according to the GFMD model.

4 Results

In this section we compare first with theoretical results in order to test the implemented solver. After this the calculated results are presented.

4.1 Comparison with theory

We are using a very simple geometry in order to compare our results with an analytical solution.

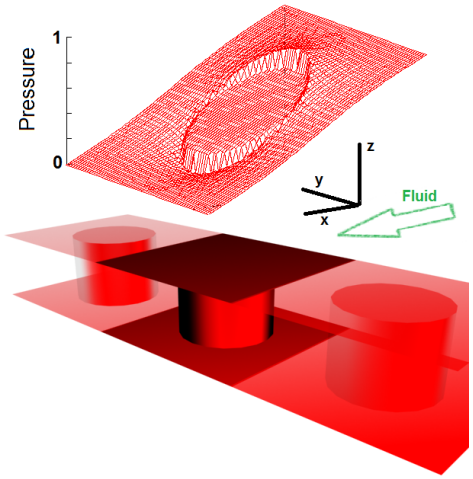


Figure 9: Simple geometry with a cylinder as contact area. The upper graph depicts the pressure course. The pressure in the middle might be appear unexpected - that happens because it is not defined inside the cylinder.

The pressure course can be intuitively understood - on the right is a slight over-pressure whereas behind the cylinder a under-pressure can be seen. The Bruggeman equation is an analytical solution with the limitation that the contact area in relationship to the whole area has to be small.

The relative difference increases with the relative contact area because of the limitation of the Bruggeman equation.

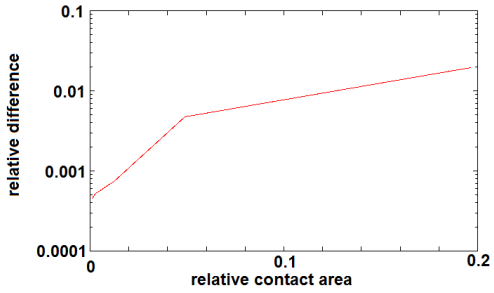


Figure 10: Relative difference between the analytical and the numerical solution in dependent of the contact area (size of the cylinder)

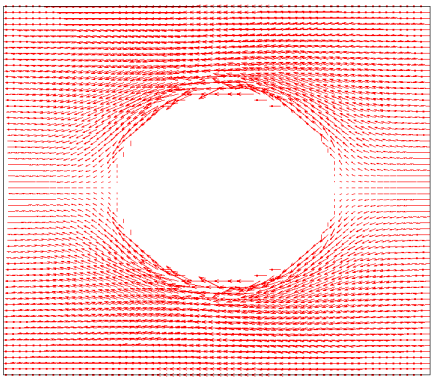


Figure 11: Calculated current along the cylinder in vector plot.

4.2 Comparison overlap model $\leftrightarrow$ GFMD

Protracting the current calculated by the solvers over the derformed surfaces by the GFMD and overlap model reveals significant differences:

In consideration of the great difference produced by the overlap model you can not justify the application of this model.

5 Conclusion

The work during the guest student programm dealed with the question “What is the flow current as a function of pressure difference and normal load with which the two surfaces are pushed together?” With an increasing normal load / contact area the current through the system decreases as depicted in figure 13 and 14.

The parallelized local solver with MPI needs about 2 minutes for a 512x512 fractal geometry on one Juropa-Node whereas the parallelized OpenMP-version needs 30 minutes on a normal PC. Unfortunately the time to compute increases highly if the system is near the critical constriction which means that there is nearly no more fluid flow through the system as a result of a strong normal load.

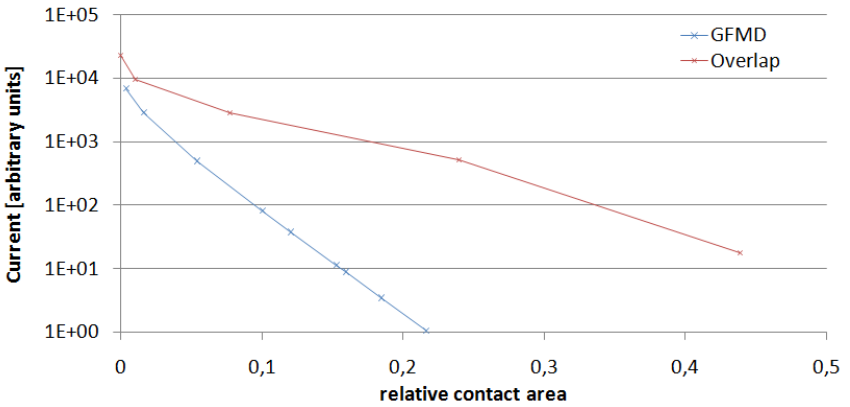


Figure 12: Calculated currents through a geometry which was deformed by the overlap or GFMD model.

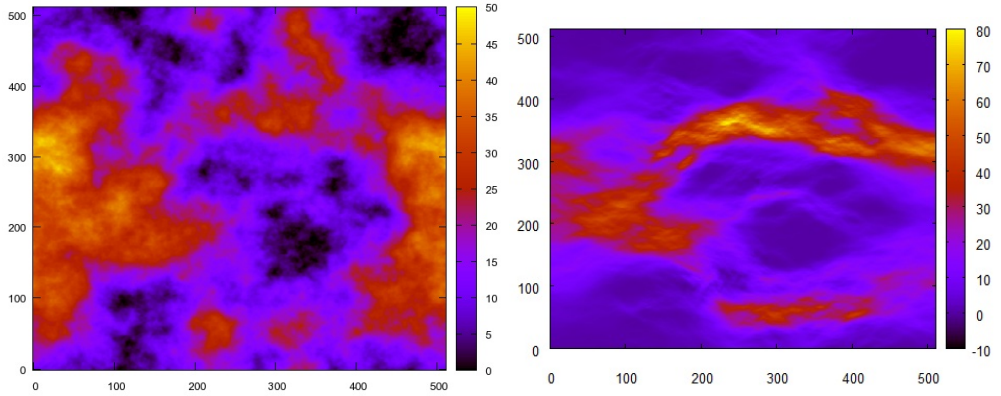


Figure 13: Left: topography with a contact area of 1.65 %. Bright areas represent a great gap whereas at dark areas the surfaces are nearly in contact. Right: Calculated current through the simulation cell. You can see two great paths for the fluid flow.

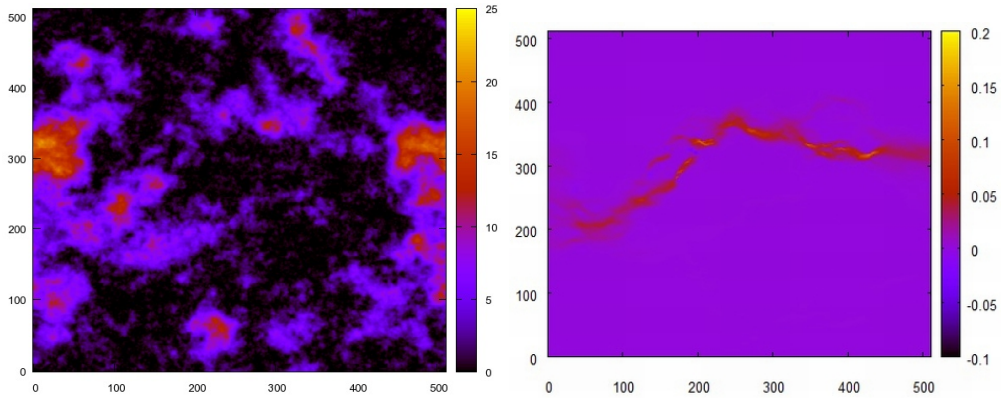


Figure 14: Left: topography with a contact area of 47.59 %. Right: Calculated current. You can see that the second path has been closed by the external force. Furthermore the current has stiffly decreased (Note the large difference in the scalebar).

Furthermore we have seen that the overlap and the GFMD model produce qualitative different results.

6 Acknowledgement

This report was written within the framework of the JSC guest student programme. Therefore I would like to thank our organisators Natalie Schröder and Mathias Winkel. They took care over a trouble-free programm and organized further events. But specially I want to thank my adviser Martin Müser who guided me during the program. With the help of his advices and support I learned a lot during this programme. Furthermore I want to acknowledge the help of Sissi de Beer and Wolf Dapp. Finally I would like to thank my girlfriend Sophie Koch for being so patient while I worked on this programme.

References

1. William Gropp, Ewing Lusk, Anthony Skjellum, MPI - Eine Einführung, Oldenbourg, 2007
2. B. N. J. Persson, J. Phys. Condens. Matter 22, 265004 (2010), section 2

GPU based visualization of Adaptive Mesh Refinement data

Francesco Piccolo

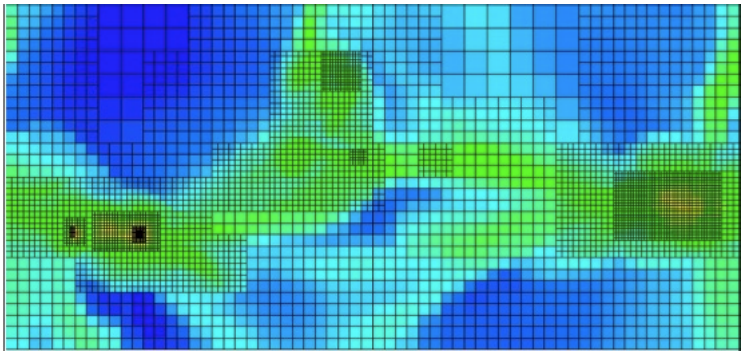
Seconda Università degli studi di Napoli
Facoltà di Ingegneria
Aversa, Italy

E-mail: piccolo.francesco@gmail.com

Abstract:
The goal of this work is a CUDA implementation of a GPU based raycasting algorithm and an octree traversal in order to speed up the visualization of AMR datasets. For this purpose complex data structures are employed to map the entire dataset to the graphics memory. An octree texture based method is used to store the data in the GPU memory and the data lookup is based on a reduced-stack traversal algorithm. The visualization algorithm uses the inherently hierarchical data structure for an efficient visualization. The volume raycasting and hierarchical data retrieval are both computationally demanding and massively parallel problems.

1 Introduction

Adaptive Mesh Refinement (AMR) is a numerical multilevel technique, associated with a particular hierarchical data structure. This technique was introduced by M. Berger in the 1980's. It is a method to discretize the continuous domain of interest into a grid of many individual elements. This method is applied in many domains like hydrodynamics, meteorology and in particular in astrophysics. In this approach relevant regions of the computational domain are represented on different levels of resolution, the related hierarchical data structure is represented by a nested rectangular subgrid. The method starts with a base coarse grid, identifies regions of interest inside the domain that require refinement and recursively adds finer and finer sub-grids until either a given maximum level of refinement is reached.



Contemporary GPUs are massively parallel processors that have outpaced CPUs in terms of floating point operations per second which makes the GPUs interesting hardware platform for solving numerous parallel compute-intensive tasks. Modern GPUs are very efficient at manipulating computer graphics, and their highly parallel structure makes them more effective than general-purpose CPUs for algorithms where processing of large blocks of data is done in parallel. Nowadays the use of hybrid (CPU-GPU) clusters is increasing due to the less power consumption per computing unit of a GPU than a CPU core.

2 GPU Architecture

CUDA is a framework for massive parallel computing on GPUs, where a hierarchy of thread groups are executed independently. Threads executed in groups are called thread block and multiple equal-shaped blocks can be executed in grids. Each thread executes the same function called a kernel. This programming model can exploit an efficient fine-grained parallelism. Threads can use more kinds of memory of different characteristics. All threads from all blocks can access one common global memory. Each thread has its own local memory that is used for some automatic kernel variables. The local memory is actually part of the global memory but each kernel can access only its own instance. Each block of threads has its own shared memory accessible by all threads of the block. Local, shared and global memories allow read-write access. There are another two types of memory available: constant memory and texture memory. Both types are read-only and both can be accessed from all threads. Texture memory also has more specific functionality like filtering and native multidimansional addressing modes. In general global memory is not cached, on the opposite the texture memory and the constant memory are cached and optimized for reading. Texture caches are designed for graphics applications where the memory access patterns exhibit a great deal of spatial locality. Texture cache is optimized for 2D spatial locality.

The G80 architecture was the first GPU which supported CUDA. It replaced separate vertex and pixel pipelines with single unified processor, that executed vertex, geometry, pixel and computing programs. In the G80 architecture each Streaming Multiprocessor(SM) contains 8 SPs (Shader Processors or CUDA Core) and 2 SFUs (Special Function Unit). This multiprocessors works on 32 threads warps and it needs 4 cycles to process a single precision floating-point operation. The stream processors share L1 and L2 cache in eight 16-shader clusters. The clusters of sixteen streaming processors also share four texture address units and eight texture filtering units, making up a total of 32 texture address units and 64 texture filtering units.

The last generation NVIDIA GPUs so called Fermi is a multiprocessor made by a scalable array of massively multi-threaded streaming processors. The SM executes threads in groups of 32 warp. Each Streaming Multiprocessor in the fermi architecture contains 32 SPs and 4 SFUs. Each SP can fulfil one single precision floating-point multiply–add (FMA) operation per cycle. Each SFU can fulfil four single precision floating point operations per cycle.

On both architectures threads in warp are free to branch independently but because multiprocessors can execute only one common instruction at a time some performance penalties occur if execution paths of threads in one warp diverge. In that case the execution of warp threads is serialized, i.e. multiprocessor executes each active path of kernel separately while threads that are not on that path are disabled. After execution of all diverged paths the execution continues again in one common path.

Graphics applications have generally a highly spatially local access to the memory, with stride patterns well known in advance (spatial locality in terms of the address space). GPU caches have traditionally been small, as in the G80 architecture, since the spatial locality means you don't need all data in the cache to service a complete memory request. Non graphics architecture like Fermi introduced non-spatially local memory access and random access patterns, which the large, unified L2 is designed to accelerate.

In the Fermi architecture each SM has a 64 KiB partitioned shared memory and L1 cache store. The cache can be partitioned two ways at the thread type level with either 16/48 or 48/16 KiB dedicated to shared memory and L1. Each sub block shares access to the store with the other, due to executing the same warp. L1 is supported by a unified L2 cache shared across each SMs.

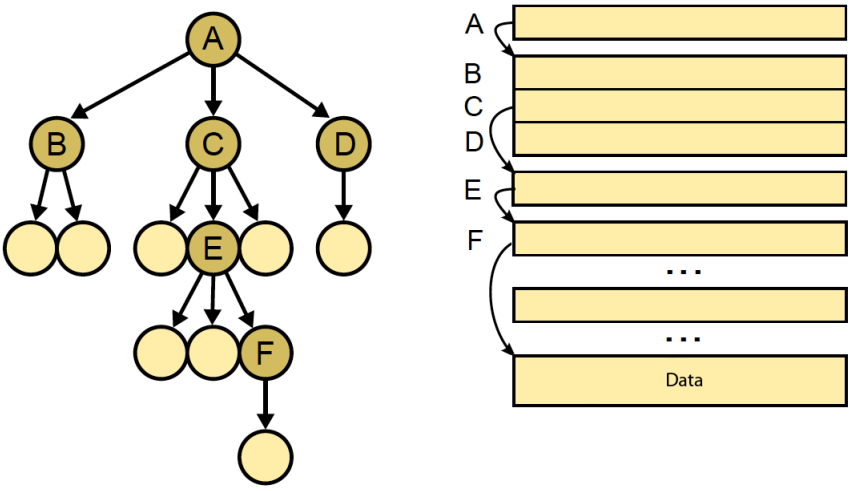
3 Octree data structure

An octree is a regular hierarchical data structure where each node subdivides the space it represents into eight octants. The first node of the tree, the root, is a cube. Each node has either eight children or no children. The eight children form a 2x2x2 regular subdivision of the parent node. A node with children is called an internal node. A node without children is called a leaf.

The tree is stored in a 3D texture called indirection pool. An indirection grid is a cube of 2x2x2 cells. Each node of the tree is represented by an indirection grid. A cell contains a data descriptor if the corresponding child is a leaf, otherwise it contains the index of an indirection grid if the corresponding child is another internal node.

The octree is stored in the texture memory instead of the global memory since there is the potential of texture caches, the addressing calculations are hidden better and no special access patterns are required to get good memory performance. There are two ways of binding data to a texture. First, data can be bound to a texture directly from the so-called linear memory, what is in fact the directly accessible global memory. Textures bound this way are restricted to be one-dimensional and no filtering or special addressing modes are supported. The second way is to allocate the data as a CUDA array. The CUDA array is a opaque memory optimized for the texture fetching that can be written from the host only. Textures bound to the CUDA array memory can be up to three-dimensional and support several filtering

and addressing modes. The use of the one dimensional addressing mode is not used because of a strong addressing memory limititation.



The data are stored in an hybrid order, all the childs of a node are in successive positions. This order simplifies the access to a node from another of the same level. From a certain node in the hierarchy it is possible to move to a brother of the current node by offset, otherwise it is possible to access to a son by an indirection address.

4 Tree lookup

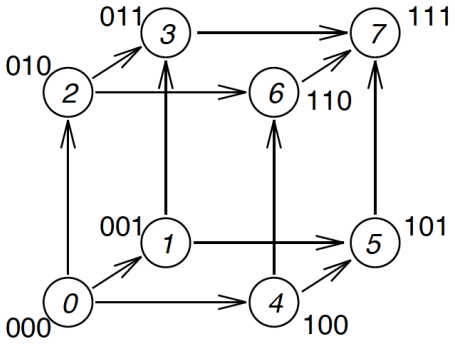
Once the data is stored in the texture memory, it is possible to retrieve the value stored in the tree at a point $M \in [0,1]^3$. The tree lookup starts from the root and successively visits the nodes containing the point M until a leaf is reached. It is easy to explain the lookup process with a quadtree.

Let ID be the index of the indirection grid of the node visited at depth D. The tree lookup is initialized with $I_0 = (0,0,0)$, which corresponds to the tree root. At depth D is necessary compute the coordinates of M within the current node.

The local coordinates of M are computed according to the formula:

$$P = floor(M + 1 - 2^{-(depth+1)})$$

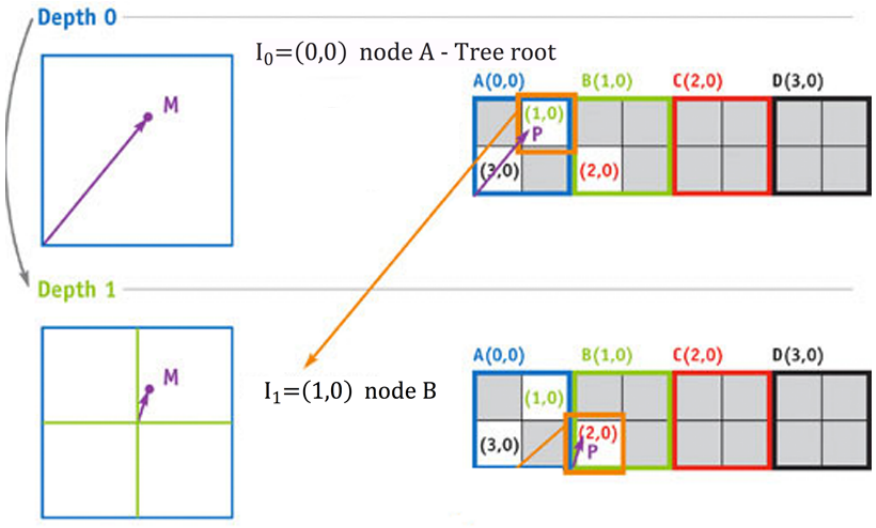
The cubes inside an octree are labelled, it is necessary to move among the elements of the current level:

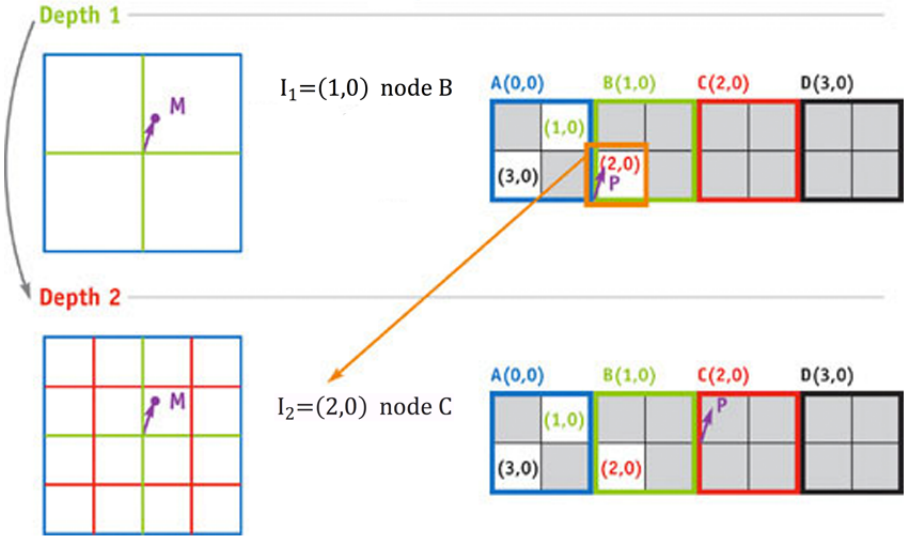


The next step is to update the indirecton grid address:

$$I_{next} = I_{prev} + P$$

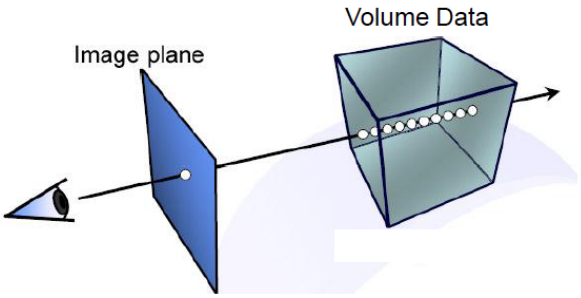
after that is possible to read the addressed indirection grid from the texture memory. At this point if a leaf is reached is necessary to calculate the data offset from the data descriptor content. The lookup process ends getting the data from the texture memory.





5 GPU Volume Raycasting

The Volume ray-casting algorithm is a direct volume rendering technique and can be derived directly from the rendering equation. This equation is an integral equation in which the equilibrium radiance leaving a point is given as the sum of emitted plus reflected radiance under a geometric optics approximation. One approach to solving the equation is based on finite element methods. The Volume Ray-casting is classified as an image-space method where the main loop is over pixels of the output image. In fact for each pixel of the image plane is traced a ray through the volume. Along the part of the ray of sight that lies within the volume, equidistant or adaptive samples are selected. Then the samples taken along the ray are composited to a single color.



There are several ways to composite the color of different elements along a ray, one of this is the so called $\alpha - compositing$.

The emission-absorption model yields a basic volume rendering equation, in terms of the radiance (power per unit area per solid angle) arriving along a ray at the position x on this ray. Start defining the optical depth as the integral of the absorption coefficient α

$$\tau(d_1, d_2) = \int_{d_1}^{d_2} \alpha(t)dt$$

and the radiation energy as :

$$C = \int_0^{\infty} c(t)e^{-\tau(0,t)}dt$$

where c(t) stands for the opacity-weighted color.

The optical depth can be approximated by the Riemann-sum :

$$\tau(0,t) = \int_0^t \alpha(t)dt \approx \sum_{i=0}^{t/\Delta t} \alpha(i\Delta t)\Delta t := \tilde{\tau}(0,t)$$

getting a first approximation of the radiance integral:

$$C = \int_0^{\infty} c(t)e^{-\tau(0,t)}dt \approx \int_0^{\infty} c(t)e^{-\tilde{\tau}(0,t)}dt =$$

$$\int_0^{\infty} c(t)e^{-\tilde{\tau}(0,t)}dt = \int_0^{\infty} c(t)\prod_{i=0}^{t/\Delta t} e^{-\alpha(i\Delta t)\Delta t}dt$$

In each voxel of the volume can be defined the opacity

$$A_i = 1 - e^{-\alpha(i\Delta t)\Delta t}$$

and the color

$$C_i = c(i\Delta t)\Delta t$$

The compositing formula is obtained from the radiation energy approximation in terms of opacity and color sampled stepping into the volume:

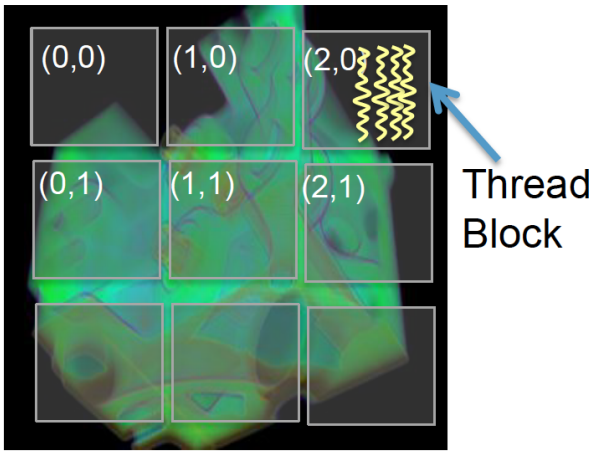
$$\tilde{C} = \sum_{i=0}^{n} C_i \prod_{j=0}^{i-1}(1 - A_i)$$

The algorithm can work front-to-back accumulating the opacity-weighted color for all the steps in the volume. In this way the computation starts with the sample nearest to the viewer and ends with the one farthest to him. This work flow direction ensures that masked parts of the volume do not affect the resulting pixel. In fact the advantage of front-to-back compositing is the early ray termination when composite transparency falls below a threshold.

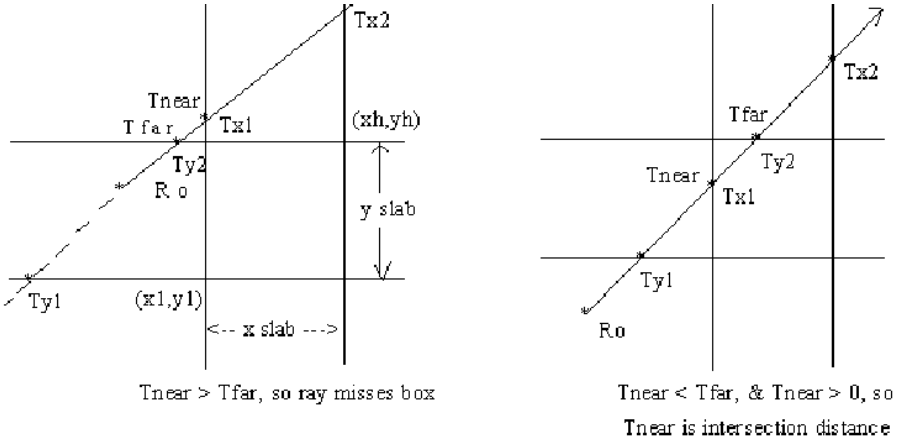
Algorithm 1 for each pixel on the image plane

```
calculate eye ray in world space
compute ray intersection with volume bounding box
while accumulatedOpacity < threshold && rayInsideBox do
    octree data lookup
    transfer function lookup
    accumulate color and opacity
    march along ray from front to back
end while
```

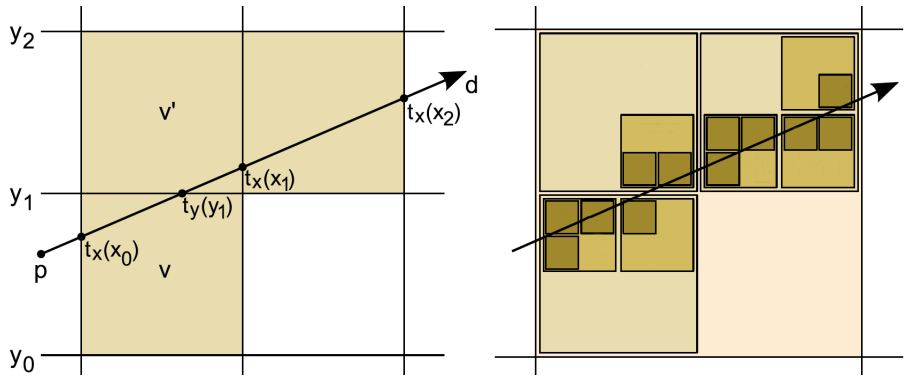
The work decomposition scheme is based on fine-grain task parallelism that achieves load balancing among the multiprocessors. In ray casting, the concurrency is obvious since we can compute each pixel value on the screen independently of all the other pixels. To take advantage of this fact, the screen is divided into a grid of small tiles. A block of threads equal to the number of pixels on each tile will be allocated for the tile and the block of threads will be executed by a multiprocessor, independently of other blocks of threads.



The Ray - Box intersection is based on the Kay and Kayjia method and it uses two pairs of parallel planes called (slabs). The method looks at the intersection of each pair of slabs by the ray, if the ray is not parallel to the plane then begin to compute the intersection distance of the planes. The intersections with the two parallel planes are Tnear and Tfar points. A ray misses the box If Tnear is greater then Tfar or if Tfar is less than zero, otherwise the entry point Tnear and exit point Tfar are calculated.



The main loop to calculate the color of each pixel consists in stepping along a ray, retrieving data sample and accumulating opacity-weighted color. The step can be fixed to the minimum resolution of the octree or adapted to the ray projected dimension of the current cube traversed. In general the adaptive sampling strategy allows to reduce the rendering time for high quality rendering dramatically, but on a SIMD multiprocessor like a modern GPU this will be only if all the threads on the same warp traverse the octree ending at the same depth. The number of steps in the volume changes pixel by pixel because of the difference of the distance beetween data's location into the octree and the lookup node, performed by different threads on the same warp.



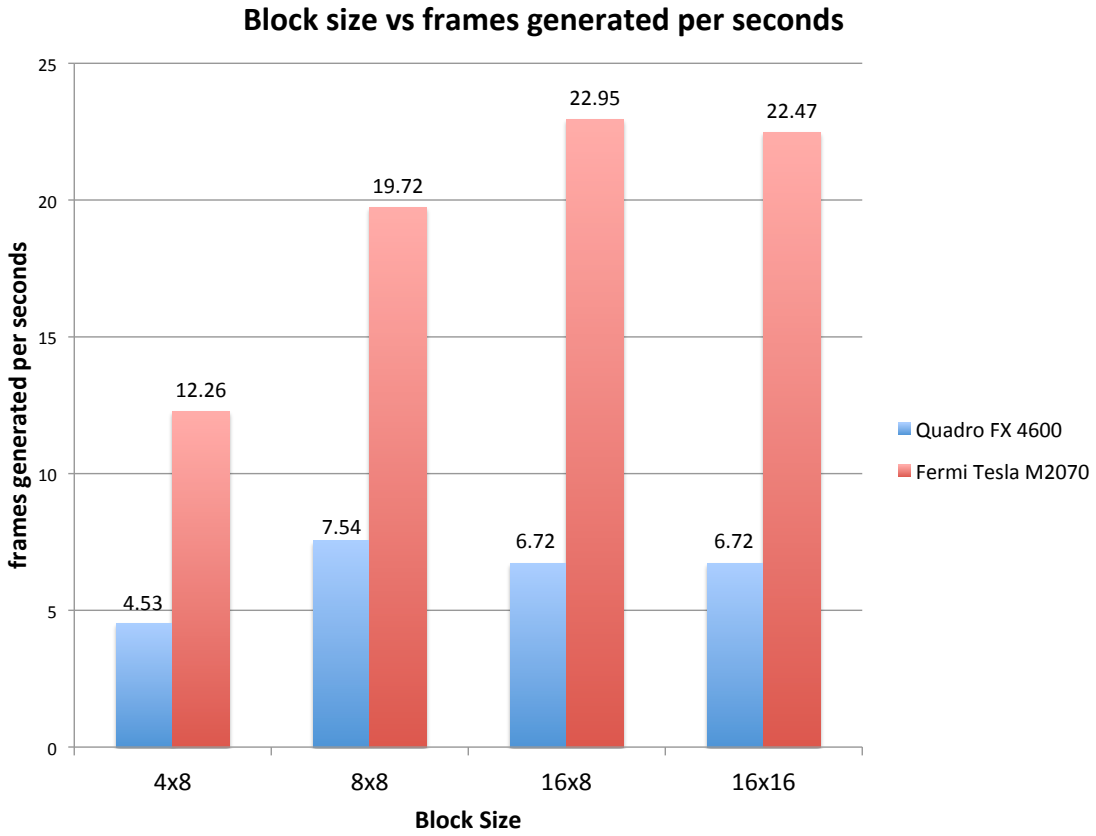
6 Results

To evaluate the performance of the visualization process, the miniJUDGE cluster has been used. MiniJUDGE is a test machine for the IBM iDataPlex GPU cluster JUDGE (JUelich Dedicated Gpu Environment). This cluster consists of 54 compute nodes, 2 login and service nodes and 2 GPFS gateway nodes. Each compute node is equipped with 2 Intel Xeon X5650 (Westmere) 6 core processors of 2.66 Ghz, 96 GB main memory and additionally 2 NVIDIA Tesla M2050 GPU (Fermi) 3 GB memory. All cluster nodes are connected via Infiniband. The test machine has the same configuration of the bigger one, but with only two nodes. Also a normal workstation has been used, equipped with one Intel Core2 Quad CPU Q9400 2.66GHz, 4GB of main memory and an NVIDIA Quadro FX 4600 graphic card.

	G80 Architecture	Fermi Architecture
	Nvidia Quadro FX 4600	Nvidia Tesla M2070
compute capability	1.0	2.0
CUDA cores	112	448
GPU Clock rate	500 MHz	1.15 GHz
Warp size	32	32
Global Memory size	777.19 Mbytes	6144 Mbytes
Memory Clock rate	700 MHz	1566.00 Mhz

The rendering target was generated using OpenGL pixelbuffer object extension [PBO], not available on the Fermi card. This extension is useful to handle OpenGL buffer objects which can be mapped into the CUDA address space and then used as global memory. In the other case the data are swapped to the main memory and diplayed as a normal pixel buffer.

The AMR dataset visualized represents data of astrophysical simulations. The dataset was built with the FLASH simulation environment and it is available in HDF5 format. It is organized in octree data, composed of ten refinement levels plus other tree levels on each leaf node. The size of the data set is 68449x8x8x8 float and the memory occupancy of the dataset on the GPU is 473.77 MByte. The measures refer to the rendering of an image plane with dimensions 512x512 pixels.



Theads inside the same block usually access data elements which are close each other in the texture memory. If the block is big different blocks computes pixels which are far away from each other, this produces more cache miss.

7 Conclusion and Future Works

This work presented an implementation of hierachical data structure on the GPU memory and a method to use this data in order to make an interactive visualization. There are several ways to extend this work. Increase the use of the texture cache converting the octree texture to a standard 2D texture, because texture memory read is optimized for a bidimensional access. The visualization quality can be improved by linear interpolation of the samples collected along the ray steps, but this will probably raise the computational time by a significative amount of time.

References

1. R. Kaehler, J. Wise, T. Abel, H.-C. Hege. GPU-Assisted Raycasting of Cosmological Adaptive Mesh Refinement Simulations. Proc. of Volume Graphics 2006, p. 103–110, Boston, USA, 2006.
2. J. E. Vollrath, T. Schafhitzel, and T. Ertl. Employing Complex GPU Data Structures for the Interactive Visualization of Adaptive Mesh Refinement Data. International Workshop on Volume Graphics (VG'06) Boston, Massachusetts, U.S.A., pp. 55-58, 2006
3. Matt Pharr, Randima Fernando. GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation, 2005.
4. A. Williams, S. Barrus, R. K. Morley, and P. Shirley. An efficient and robust ray-box intersection algorithm. Journal of Graphics Tools: JGT, 10(1) p. 49–54, 2005.
5. D. Kirk, W. Wen-mei Hwu: Programming Massively Parallel Processors, A Hands on Approach. Morgan Kaufmann, 2010.
6. S. Laine, T. Karras. Efficient Sparse Voxel Octrees. IEEE Transactions on Visualization and Computer Graphics 17, 8, p. 1048-1059, 2011.
7. FLASH User's Guide.
8. NVIDIA CUDA Programming Guide from the CUDA Toolkit v4.0.
9. NVIDIA CUDA Best Practices Guide from the CUDA Toolkit v4.0.
10. NVIDIA CUDA Reference Manual from the CUDA Toolkit v4.0.
11. NVIDIA's Next Generation CUDA Compute Architecture: Fermi. NVIDIA Fermi Compute Architecture Whitepaper.
12. NVIDIA Tuning CUDA Applications for Fermi v1.0.

Brain volume reconstruction - parallel implementation of unimodal registration

Petar Sirković

Faculty of Science, Mathematics department
University of Zagreb
Bijenička 30
10000 Zagreb
Croatia

E-mail: petar.felixx@gmail.com

Abstract:
In order to construct a 3D brain volume, a large number of 2D brain slice images are combined. These images are usually significantly deformed during the preparation process and they have to be mapped to the correct geometrical place. In cases when there is no geometrically correct reference image, a sequence of mappings between neighbouring images is produced. This process is usually strictly sequential. It takes several hours to compute one binary registration, which leads to several months needed to produce a full brain volume. The purpose of this work is to test the parallelisation of this process, the concatenation of binary registrations and investigate the potential problems that arise on the way.

1 Introduction

Within the human brain mapping project at the research centre Jülich, one of the main tasks is to create a high resolution 3D brain image. A large number ($\sim$ 1200) of 2D high resolution brain slice images is used to construct it. These images are usually obtained using PLI[1] technique which reveals the spatial orientations of nerve fibres. This technique relies on the post-mortem brain histology. In the process of making these images, brain needs to be cut into very thin slices which are then mounted on glass slides. This procedure inevitably introduces some deformations such as local shearing and tearing.

[1] PLI - Polarized light imaging

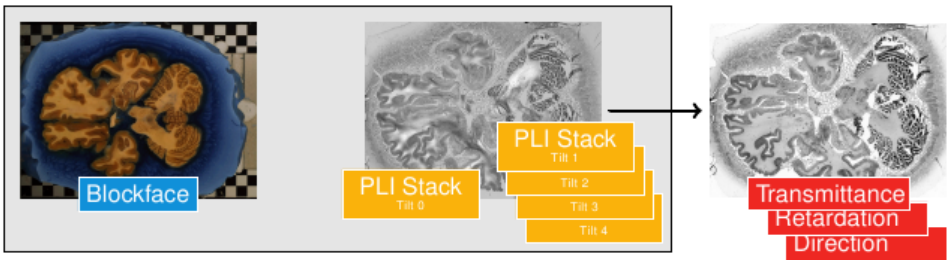


Figure 1: Images from different modalities.

The original geometrical 3D shape is restored by mapping the slice images onto some geometrically correct reference images. Usually these reference images are less detailed, and in some cases they are not available at all. Then, neighbouring slices are taken as reference. Series of binary registrations between the neighbouring slices are performed. These programs are usually strictly sequential. First, slice no. 2 is mapped onto slice no. 1, then slice no. 3 is mapped onto new slice no. 2, etc. The problem with this approach is that the since images have high resolution (usually 10 megapixels or more), one binary registration takes about 3 hours on one core of a modern computer. Taking this into account, it is easy to compute that it takes about 150 days to perform this process for the whole brain.

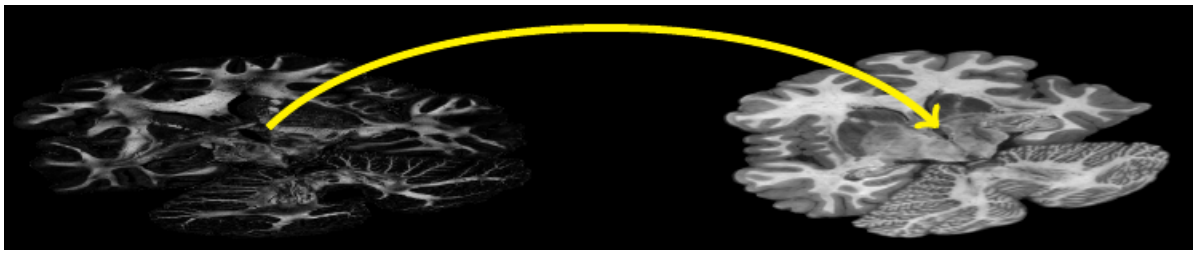


Figure 2: Preview of image mapping. The task is to align two slices of possibly different image modalities, showing the same tissue parts under linear and nonlinear distortions.

1.1 Ideas for improvement

Instead of performing these registrations sequentially, binary registrations between original slice images could be performed independently and a full mapping for one particular slice image can then be obtained by performing a concatenation of the results of the binary registrations. This concatenation of the consecutive mappings requires an interpolation whose results should also be checked. The process time can be further reduced by dividing images into parts (with some overlap) and performing registration between parts of neighbouring slice images almost independently, needing only some stitching of the edges of the image parts. In the further text, a mapping process between images will be called a registration, and its result will be called a deformation field.

2 Theoretic part

2.1 Some measures for measuring registration quality

Normalized correlation [1]
Normalized correlation (NC) is a similarity measure between two unimodal images. Given a fixed image f, and a moving image m, it is defined by this formula:

$$\text{NC}(f,m) = \frac{\sum_{i=0}^{N-1} d(f_i)d(m_i)}{\sqrt{(\sum_{i=0}^{N-1} d(f_i))(\sum_{i=0}^{N-1} d(m_i))}},$$

where i loops over all pixels, $d(f_i) = f(p_i) - \mu(f)$ and $d(m_i) = m(p_i) - \mu(m)$, and $\mu(f)$ and $\mu(m)$ are mean values of the fixed and moving images intensities in the overlapping region, respectively.

Normalized mutual information [2]
Normalized mutual information (NMI) is a multimodality similarity measure. It is defined by this formula:

$$\text{NMI}(f,m) = \frac{H(f) + H(m)}{H(f,m)},$$

where f and m are fixed and moving image, respectively. $H(f)$ and $H(m)$ denote the single image entropies

$$H(f) = \sum_{w^f} p(w^f) \log_2 p(w^f)$$

$$\text{and } H(m) = \sum_{w^m} p(w^m) \log_2 p(w^f),$$

and $H(f,m)$ denotes the joint entropy of the images

$$H(f,m) = \sum_{w^f, w^m} p(w^f, w^m) \log_2 p(w^f, w^m).$$

Dice coefficient [3]
Dice coefficient (DC) is a simple measure which measures the overlap of the regions (tissues) of interest and is defined by

$$\text{DC}(a,b) = 1 - \frac{N(|a-b|)}{N(a) + N(b)},$$

where $N(a)$ and $N(b)$ are the numbers of pixels in regions of interest in the respective images, and $N(|a-b|)$ is number of the non-overlapping pixels.

2.2 Registration methods

Registration is the process of mapping the moving image to the fixed image. It is usually performed as an iterative optimization process made up from 3 stages: applying a transform to the moving

image, interpolating it to the grid points and evaluating the quality of the transform by some measure.

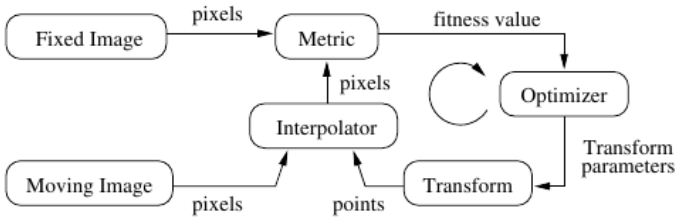


Figure 3: Registration process scheme.

Rigid registration[1]
Rigid registration is one of the simplest registration types. It uses rigid transformations which are defined by

$$T_r(p) = \mathbf{A}_r p + \mathbf{t} = \begin{pmatrix} \cos\varphi & \sin\varphi \\ \sin -\varphi & \cos\varphi \end{pmatrix} p + \begin{pmatrix} t_x \\ t_y \end{pmatrix},$$

where p represents a pixel position. The transformation is basically a composition of a rotation by the angle φ and a translation $\mathbf{t}$. Values for these parameters are optimized with respect to some measure introduced before.

Affine registration[1]
Affine registration is the general linear registration. It uses affine transformations which are defined by

$$T_{af}(p) = \mathbf{A}_{af} p + \mathbf{t} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} p + \begin{pmatrix} t_x \\ t_y \end{pmatrix},$$

where p represents a pixel position. The transformation is a composition of a linear mapping in 2D (matrix $\mathbf{A}_{af}$) and a translation $\mathbf{t}$. Values for these parameters are optimized with respect to some measure introduced before.

2.3 Nonlinear registrations

In order to find a true shape for an image which has been deformed by some non-linear deformation non-linear registration methods must be used. In this work the demons registration algorithm [4] [5] was used, but there are some other methods such as free-form-deformation that uses B-splines, elastic and fluid registration [1].

Demons registration [5] [4]
In the demons registration method voxels in the static image generate local forces to displace voxels in the moving image. The moving image is iteratively deformed. To each voxel a displacement vector $d\mathbf{r} = (dx, dy, dz)$ is applied. The displacement vector in the n-th iteration is computed as

$$d\mathbf{r}^{n+1} = \frac{(I_m^{(n)} - I_s^{(0)})\nabla I_s^{(0)}}{(I_m^{(n)} - I_s^{(0)})^2 + (\nabla I_s^{(0)})^2},$$

where $I_s^{(n)}$ and $I_m^{(n)}$ are the intensity of the static and the moving image at the n-th iteration, respectively. They are computed with the aim to reduce value of some fuction of local differences between the fixed and the moving image. There are different versions of the equation for the displacement vector. This version is convenient because the gradient of the static image $\nabla I_s^{(0)}$ is constant through all the iterations and thus it has to be computed only once [5]. This version is, because of that, called passive force method. The important thing to notice is that in the demons registration method voxels move independently and because of that there is no guarantee for the smoothness of the result [4].

3 Implementation

In this work a parallel version (algorithm 1) of the reconstruction of 3D brain image is implemented. Binary registrations between neighbouring slices are performed in parallel. Demons registration method was used for the registration calculation, in particular, the ITK implementation of the method. Calculated deformation fields are then concatenated using bilinear interpolation. For the communication between processes, the MPI[2] [6] interface is used. Processes handle their inputs and outputs independently.

Algorithm 1 Pseudo code

```
MASTER DO read problem size
MASTER DO distribute jobs to other processes
ALL DO read your images
ALL DO perform your registrations
ALL DO perform concatenations of your own registrations
ALL DO perform "Allconcatenate" - processes exchange data
ALL DO apply new concatenations to all your deformation fields
ALL DO apply deformation fields to images and print them out
```

Insight Segmentation and Registration Toolkit [7]
Insight Segmentation and Registration Toolkit (ITK) is an open-source, cross-platform system that provides an extensive suite of software tools for image analysis. In this work the ITK implementation of demons registration method *itkDemonsRegistrationFilter* was used with the following parameters: number of iterations $= 1500$, standard deviation $= 1$. The deformation field $d_{B,A}$ produced by the method satisfies $B'(p) = B(p + d_{B,A}(p))$, where B' is image B registrated on the image A. For applying a deformation field to an image the ITK method *itkWarpImageFilter* was used.

3.1 Concatenation of deformation fields

After applying one deformation field, mapped images don't end up on the grid of the other image. Concatenation of the deformation fields is simply an addition of movements described in the fields. In order to concatenate them, interpolation of the deformation fields from grid points to all points

[2] MPI - Message Passing Interface

in the plane is needed. The number of the concatenations of non-linear deformation fields that is reasonable to make depends on the smoothness of the fields. In this work bilinear interpolation is implemented. Coefficients for the interpolation are inversely proportional to the distance of the 4 surrounding grid positions. Concatenations of the deformation fields are also done in parallel. Firstly, processes concatenate all deformation fields calculated by them. After that bottom-up accumulation (algorithm 13) of data is performed.

Algorithm 2 Communication scheme - Allconcatenate

```
ALL DO add ← 0
ALL DO current ← my concatenated deformation fields
while step < nproc do
    neighbour ← myrank ⊕ step
    MPIsendrecv(current, tmp, neighbour)
    if myrank ∧ step = 0 then
        current ← concatenation(current, tmp)
    else
        current ← concatenation(tmp, current)
        current ← concatenation(tmp, add)
    end if
    step ← step ∗ 2
end while
```

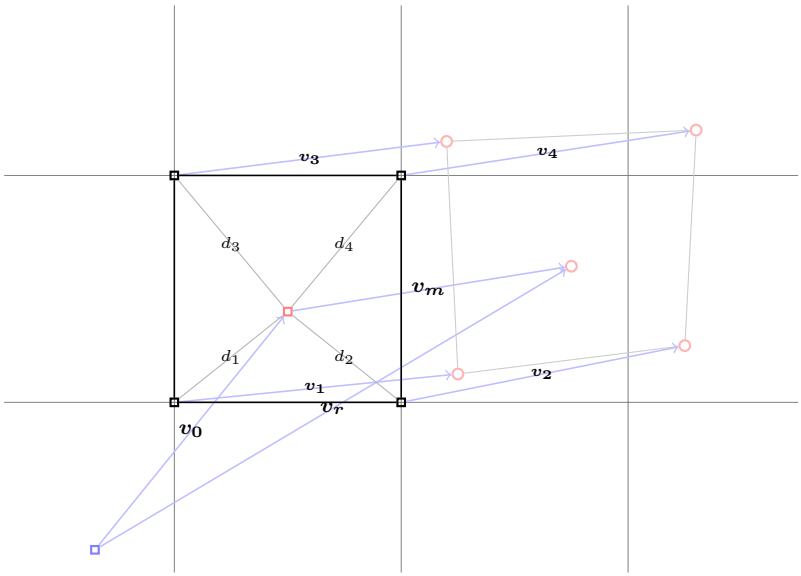


Figure 4: The blue square is a pixel position in image no. 2 and it is mapped by vector $\boldsymbol{v_0}$ onto image no. 1. The black squares are grid positions in the image no. 1, and they are mapped to the image no. 0 by vectors $\boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3}$ and $\boldsymbol{v_4}$. We would like to compute mapping of the original pixel from image no. 2 to the image no. 0. We use interpolation of transformation from image no. 1 to image no. 0 to calculate the result. The resulting vector of the concatenation is obtained as a linear combination, $\boldsymbol{v_r} = \boldsymbol{v_0} + \boldsymbol{v_m} = \boldsymbol{v_0} + \frac{\sum_{i=1}^{4} \frac{\boldsymbol{v_i}}{d_i}}{\sum_{i=1}^{4} \frac{1}{d_i}}$.

4 Results

4.1 Testing

One node (12 cores) of the JUDGE supercomputer was used. Only CPUs were used for the computation because ITK works only on CPUs. One node of the JUDGE contains two Intel Xeon X5650 (Westmere) 6-core 2,66 GHz processors. Testing was done on the images whose sizes are 5%, 10%, 25% or 50% of the full brain image size.

4.2 Result preview

Results of the algorithm (figure 5) over a short number (~ 4) of concatenations are ok. There is not much blurring and almost no additional artifacts arise, but after more concatenations (~ 32) the resulting image is very blurry and it looks like someone has shaken it a bit.

4.3 Scalability

The registration part of the algorithm has a perfect time scaling, which was expected since the registrations are computed completely independently. However, the concatenation part of the algorithm doesn't scale that perfectly, because of the $O(\log m)$ factor that was introduced in the tree-like communication scheme. Results can be seen in figure 6.

4.4 Load balance problems

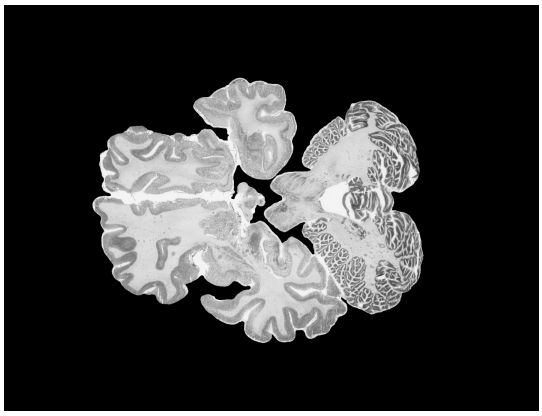
The distribution of registration times for the different neighbouring slices was analyzed in table 1 in order to see if there are some load balancing problems and to potentially eliminate them. The analysis shows that the registration times don't differ very much (max $\sim 1\% - 2\%$) from each other, and we can conclude that there are no load balancing problems.

Img size	Avg time	Std dev time	Min time	Max time
5%	9.743875	0.058692	9.675507	9.894207
10%	33.262259	0.133462	33.036071	33.591253
25%	213.756354	1.175624	212.468790	216.644891
50%	1046.990324	6.948083	1038.305198	1064.010131

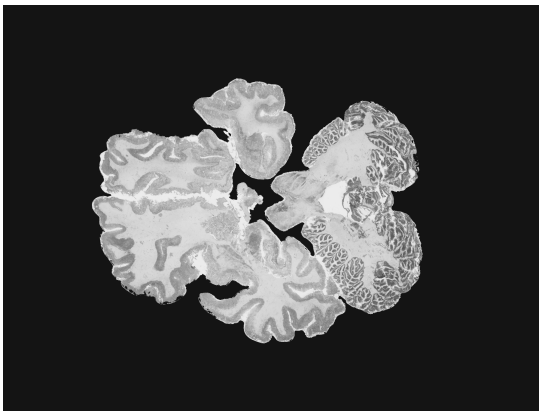
Table 1: Analysis of the binary registration times and the load balance problems.

4.5 Statistical analysis of the section movements

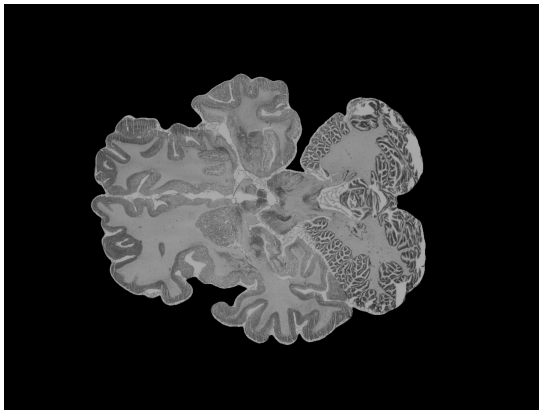
The movement of the pixels over more concatenations was analyzed in the figure 7 in order to see what are the potential problems in the method and how it can be speeded up even more. The average pixel movement grows approximately linearly with the number of concatenations, and the maxi-



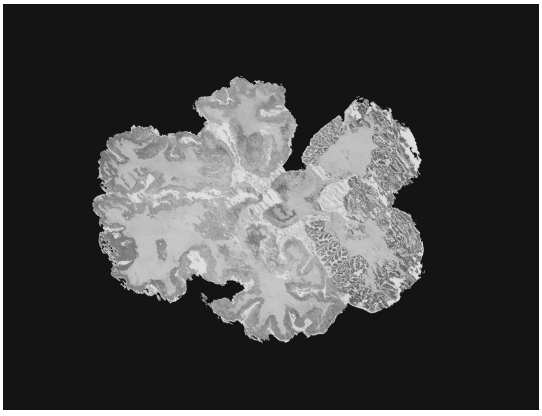
(a) Original image no. 4



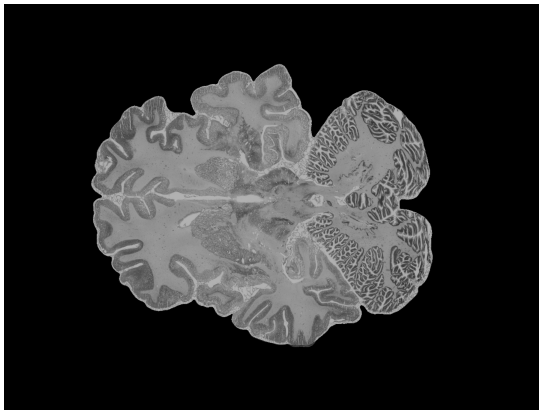
(b) Moved image no. 4



(c) Original image no. 16



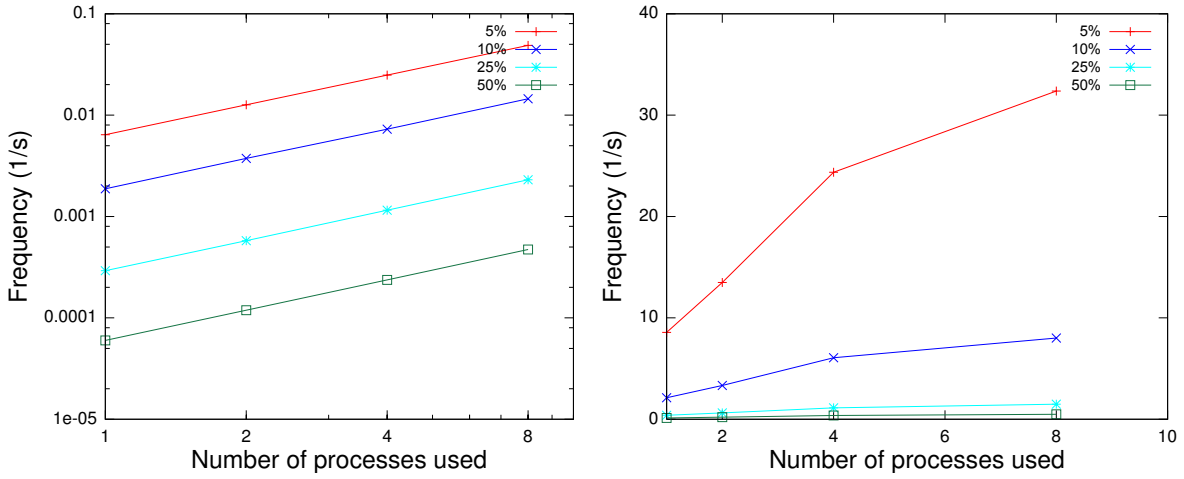
(d) Moved image no. 16



(e) Original image no. 64

(f) Moved image no. 64

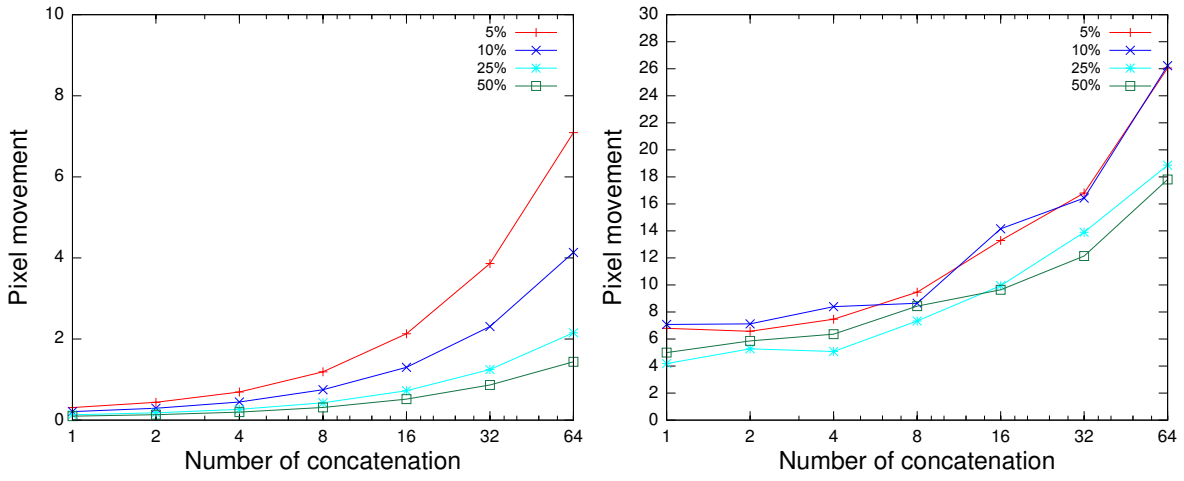
Figure 5: On the left side are the original images 4, 16 and 64. On the right side are the same respective images after applying 4, 16 and 64 concatenated deformation fields, respectively. Before calculating the registrations, histogram matching of the images was applied and this causes the differencies in the brightness of images. It can be seen that the number of the arfifacts that arise on the result images grows with the number of slices the concatenations are bridging.



(a) registration time (b) concatenation time

Figure 6: The left picture shows the scaling of the inverse of the registration time with the increasing number of processors. Expected scaling is O(n). The right picture shows the scaling of the inverse of the concatenation time with the increasing number of processors.

mum pixel movement after the initial increase also grows linearly with the number of concatenations.



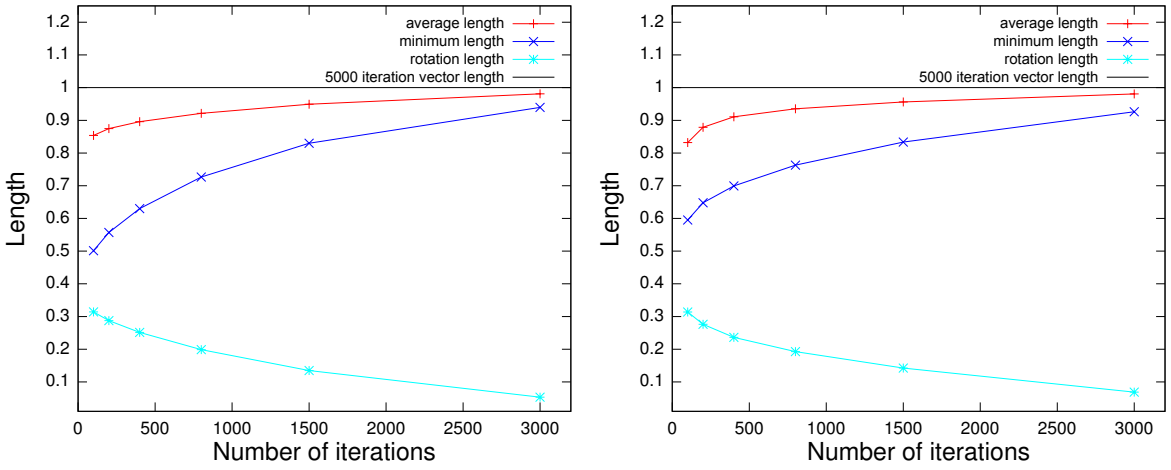
(a) average (b) maximum

Figure 7: Average and maximum pixel movement over specified number of concatenations.

4.6 Analysis of the number of iterations needed

Problems with the convergence of the demons registration method were spotted while testing some simple test cases. That's why the length of the vector produced by the method with different number of iterations was analyzed. After the analysis in the figure 8 we can see that already after 100 iterations,

almost full length has been achieved. After that the changes in the length are very small, and it's not clear whether the method converges or not. Also, since the graphs for the 5% and 10% images look very similar, the convergence problem shouldn't depend on image size.



(a) 5% images

(b) 10% images

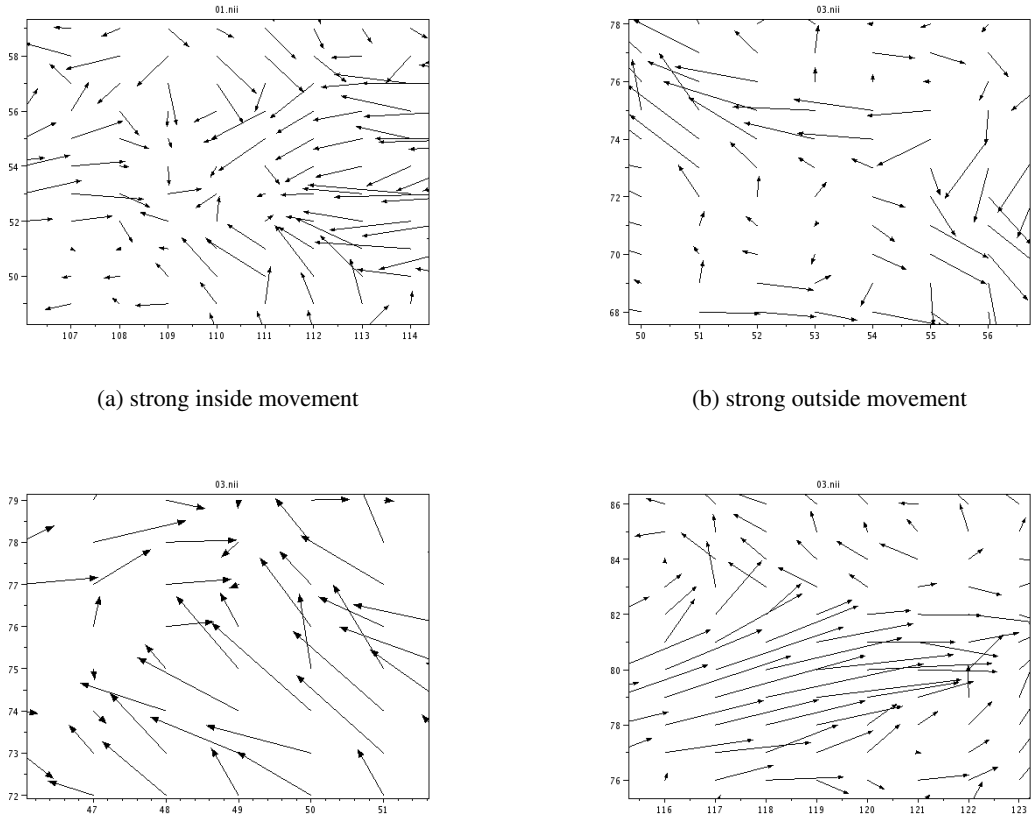
Figure 8: Analysis of the deformation vector length concerning different number of iterations. The numbers are scaled with the length of the deformation vectors after 5000 iterations.

4.7 Remarks

The transformation field produced by the ITK method is not smooth. It is not reasonable to expect that the interpolation of such fields would produce useful results. Diffusion smoothing was also implemented but it was not clear if it provides better results, since without real reference, there was no other way to say whether the results are good or not, except optically. Interpolation of transformation fields with polynomials up to the power of 2 was tried, but it did not provide useful results. In the figure 9, there are examples of a few images where we can see parts of the deformation fields produced by the ITK method, which include strong inside movement towards to or strong outside movement outside of one pixel. Also, there are parts where the deformation field is not smooth, or not even monotonous.

5 Conclusion

The goal of this work is fulfilled. The method is fully parallelized with almost perfect scaling. It can be concluded that there are almost no load balancing problems, since the scaling is almost perfect, and the registration times are all in few percent of the average time. The results it provides are ok up to a small number of concatenations. After more concatenations it doesn't produce good results, probably because of the non-smoothness of the initial deformation fields. Problems with the ITK implementation of the demons registration method were spotted. It does not produce smooth deformation fields and also includes a lot of local rotations. It should be further analysed if the method converges and when,



(a) strong inside movement

(b) strong outside movement

(c) non-monotonous movement

(d) non-smooth movement

Figure 9: Preview images of some specific regions of binary registrations where some unwanted artifacts arise.

since from the tests made it is not clear. From the results can also be concluded that the concatenation time is significantly smaller than the registration time, and that the registration part of the algorithm is a bottleneck. A potential solution for this problem can be section-wise registration. In order to enable this pixel movements over a sequence of registrations were investigated, since it is a good starting point for the section-wise registration (estimate for the needed overlap). Additionally, this method can be tested with some other registration method that produces more smooth deformation fields. It shouldn't be a lot of work since the concatenation part of the algorithm is independent from the registration part.

Acknowledgements

I would like to thank my advisors Dr. Bernhard Steffen, Oliver Bücker (Jülich Supercomputing Centre, Research Centre Jülich) and Dr. Timo Dickscheid (Institute of Neuroscience and Medicine, Research Centre Jülich) for their help and understanding while doing this project.

References

1. Christoph Palm, Markus Axer, David Graessel, Juergen Dammers, Johannes Lindenmeyer, Karl Zilles, Uwe Pietrzyk, and Katrin Amunts, Towards ultra high resolution fibre tract mapping of the human brain - registration of polarised light images and reorientation of fibre vectors, Frontiers in Human Neuroscience, (2010).
2. Colin Studholme, Derek L. G. Hill, David J. Hawkes, An overlap invariant entropy measure of 3D medical image alignment, Pattern Recognition, Vol. 32, Jan 1999
3. Lee R. Dice, Measures of the Amount of Ecologic Association Between Species. Ecology 26, 1945.
4. Jean Philippe Thirion, INRIA, France, Image matching as a diffusion process: an analogy with Maxwells demons, Medical Image Analysis, 1998.
5. Xuejun Gu, Hubert Pan, Yun Liang, Richard Castillo, Deshan Yang, Dongju Choi, Edward Castillo, Amitava Majumdar, Thomas Guerrero and Steve B Jiang, Implementation and evaluation of various demons deformable image registration algorithms on a GPU, Physics in medicine and biology, 11.12.2009.
6. MPI documenatation, URL: http://www.open-mpi.org/doc/
7. ITK software guide, URL: http://www.itk.org/ItkSoftwareGuide.pdf

Porting and optimization of a Lattice Boltzmann D2Q37 code to Blue Gene/Q

Fabio Pozzati

University of Bologna

E-mail: pozzati@fe.infn.it

Abstract:
We describe the implementation and optimization of a Lattice Boltzmann code for computational fluid-dynamics on the massively parallel BlueGene/Q architecture. We analyze the behaviour and the performance using a prototype version of BG/Q which is installed at the IBM research lab Böblingen. Using the large degree of parallelism of the underlying algorithm, it is possible to make use of all the available parallel resources of the architecture (multi-node, multi-core, SIMD).

1 Introduction

Fluid-dynamics is studied today with the support of computational techniques for the highly non-linear equations of motion, in regimes interesting for physics or engineering.
Over the years, many different numerical approaches have been proposed and implemented on several massively parallel computers.
The Lattice Boltzmann (LB) method is a flexible approach, able to cope with many different fluid equations (e.g., multiphase, multicomponent and thermal fluids) and to consider complex geometries or boundary conditions.
LB then describes on the computer some simple synthetic dynamics of fictitious particles ("populations") that evolve explicitly in time and, appropriately averaged, provide the correct values of the macroscopic quantities of the flow. The main advantage of LB schemes from the computational point of view is that they are "local" (they do not require the computation of non local fields, communications are only amongst nearest neighbors).
In recent years, the level of parallelism on a single processing node has increased, e.g. due to a larger number of cores and wider SIMD vector units.
The challenge now is to combine effectively inter-node and intra-node parallelism.
In this report, I present the implementation of an LB application on an IBM BlueGene/Q (BG/Q) prototype, a massively parallel supercomputing system, based on a multi-core processor.
The main tasks we had to address were two; i) adapting a complex numerical algorithm to the new architecture and ii) split the computation on a set of different processing elements. The LB algorithm that

we consider here is described in [5, 6]; it has already been ported and optimized onto several massively parallel system as described in details in [2, 3, 4]

2 Lattice Boltzmann Method

The Lattice Boltzmann methods (LBM) is a class of computational fluid dynamics (CFD) methods. This computational method is a simulation of synthetic dynamics described by the discrete Boltzmann equation, instead of the Navier-Stokes equations.
The lattice description is given in terms of an LB discretization ($f_l(x,t)$ are the lattice populations):

$$f_l(\mathbf{x}+\mathbf{c}_l\Delta t, t+\Delta t) - f_l(\mathbf{x},t) = -\frac{\Delta t}{\tau}\left(f_l(\mathbf{x},t) - f_l^{(eq)}\right),$$

where the equilibrium distributions $f_l^{(eq)}$ is defined in terms of the macroscopic hydrodynamical fields. The main idea is that the virtual particles interact in two phases, `streaming` and `collision`, which reproduce the dynamics of fluids after appropriate averaging.
Since this method from the computational point of view is local, it can be parallelized on a large number of nodes/cores with good efficiency. In this work we consider the D2Q37 model which is suitable to study behaviour of compressible gas and fluids.
The D2Q37 model (2-dimensions and 37 populations) is a model used to study accurately the "Rayleigh-Taylor instability", an interface instability of two fluids of different densities triggered by gravity.
For instance, a cold-dense fluid over a less dense and warmer fluid triggers an instability that leads to mixing of the two regions until the equilibrium is reached. For more informations see [1].

3 D2Q37 implementation

Here we describe the structure of the LB code, the data-structures and the implementation details.
To represent the fluid, we need a data-structure which comprises the 37 double-precision floating point values representing the population of a fluid-cell in the D2Q37 model.
In this case we use a data-structure called `pop_type` that contains the array of populations and some other parameters (velocity, temperature, etc.).
At each time step, each lattice-site is processed by applying three main phases: `stream, bc` and `collide`.

```
typedef struct {
  double p[37];  // populations array
  double u;      // horizontal velocity
  double v;      // vertical velocity
  double rho;    // density
  double temp;   // temperature
  ...
} pop_type __attribute__ ((aligned (32)));
```

```
void init(f) {
  for ( i = 0 ; i < NSITE ; i++ )
    for ( k = 0 ; k < 37 ; k++ )
      f[i].p[k] = initialPopValue;
}

void LBM_funct() {
  pop_type f1[NX*NY], f2[NX*NY];
  for ( step = 0 ; step < MAXSTEP ; step++ ) {
    stream (f1, f2);
    bc (f1, f2);
    collide (f2, f1);
  }
}
```

- `stream()`: this phase gathers for each site the populations according to the scheme shown in Fig. 1. This process does not perform any floating-point computations but only accesses sparse blocks of memory locations. This phase requires to access all neighbor-cells at distance 1, 2 and 3 within the grid that will collide during the next computational phase (`collide()`).
- `bc()`: this phase sets the boundary conditions, i.e. adjusts values of the cells at the top and bottom boundaries of the lattice as shown in Fig. 2 (e.g., a constant given temperature and zero velocity).
- `collide()`: this phase performs all the mathematical steps needed to compute the new population values at each lattice site. This is the floating point intensive part of the code with ≈ 7820 DP operations per site. This phase is completely local, in fact it uses only the population of the site on which it operates.

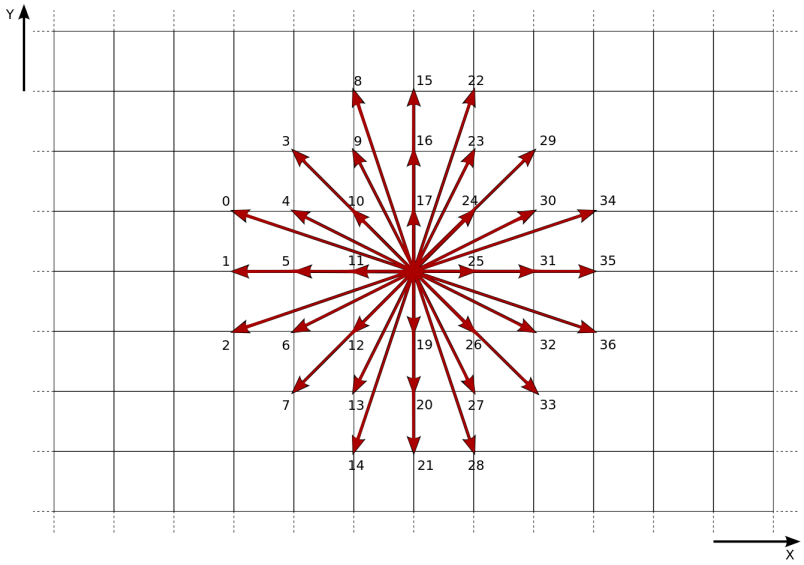


Figure 1: Visualization of the stream phase. Populations at a distance 1, 2, 3 lattice-points are gathered to the lattice-point at the center.

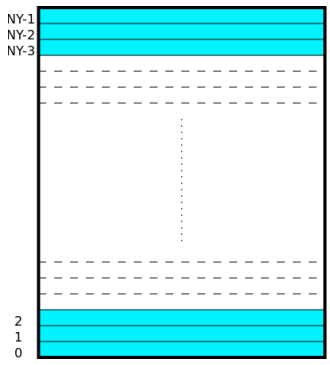


Figure 2: The bc phase needs to adjust the values only in the upper and lower 3 rows of the lattice.

To make the simulation, we maintain two copies of the lattice, each phase reads the inputs from one lattice and writes the results to the other.

4 Parallelization

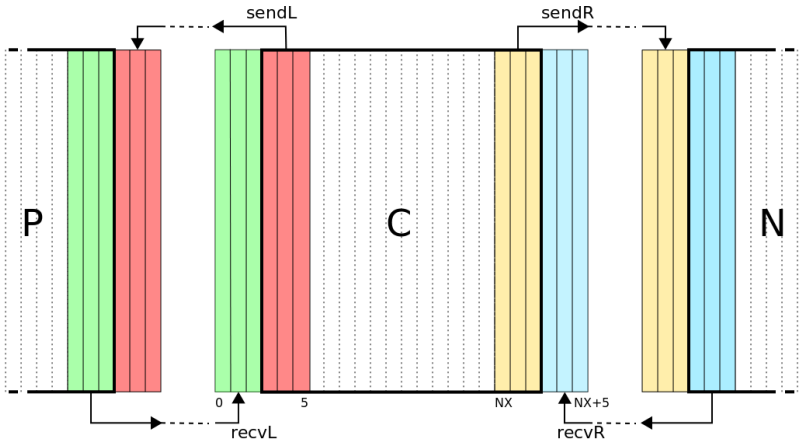


Figure 3: The first 3 columns of the sub-lattice of process C are sent to process P, the last 3 columns are sent to process N and process C receives data from N and P.

This code, as said before, is massively parallelizable because of its locality properties.
Now we explain all the strategies applied to the code to exploit all the parallelism available in the LBM algorithm both intra-node and inter-node.
First we consider the parallelization over multiple nodes. Here we use **MPI** to divide the job over several **MPI-processes**. A lattice of size $L_x \times L_y$ is split over N_p processes along X direction each processing a sub-lattice of size $\frac{L_x}{N_p} \times L_y$. During the stream() phase we need the neighbor-cells at distance 1, 2 and 3 in X direction. The cells close to the Y borders of the sub-lattice of each process need data of the neighboring sub-lattices processed by two other processes.
Therefore, before entering the stream() phase we have to exchange the first and the last 3 columns of the sub-lattice with the neighbors. To do this we need a new phase called **comm()** where we exchange the data between the processes, see Fig. 3.

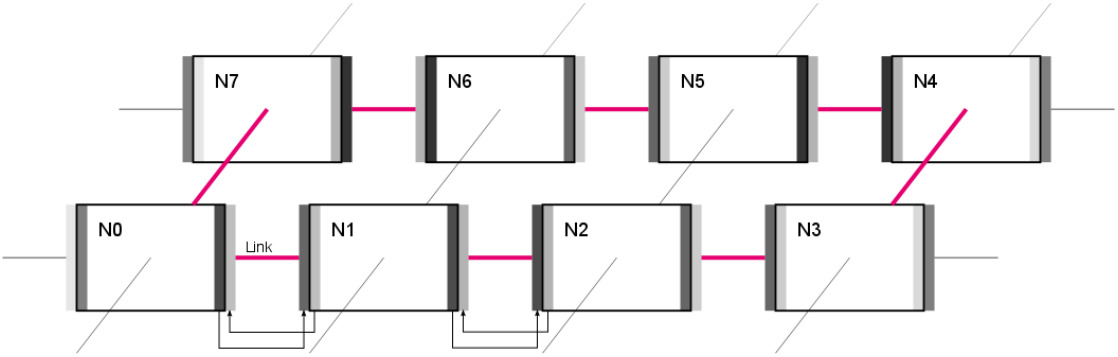


Figure 4: Mapping of the application onto a 2 dimension torus/mesh network.

This schema implies an ordering of the nodes along a virtual ring, so each node is connected with a previous and a next node as shown in Fig. 4. A first optimization which we implemented in order to make use of the on-chip parallelism is to implement a thread parallelization inside a single MPI-process.
For this we use the standard **pthread** library. This approach avoids any overheads observed for other high level libraries. Using the threads, we can further divide the sub-lattice over each thread and compute each thread-lattice concurrently. If we have a sub-lattice of size $\frac{L_x}{N_p} \times L_y$ and N_t threads, each thread has to compute only a thread-lattice of size $\frac{L_x}{N_p N_t} \times L_y$. This optimization often leads to a big increase of performance, especially on multicore architectures.
In Fig. 5 we can see the splitting of the lattice between the threads. We now have a closer look into the code executed by each process. For now we use barriers to synchronize between the phases of the algorithm. (As discussed later it is possible to overlap processing of different phases.)

```
void LBM_funct() {
  for ( step = 0; step < MAXSTEP; step++ ) {
    if ( tid == 0 ) {
      comm();     // exchange borders
    }
    pthread_barrier_wait(...);

    stream();     // apply stream()
    pthread_barrier_wait(...);

    if ( tid == 0 ) {
      bc();       // apply bc()
    }
    pthread_barrier_wait(...);

    collide();    // compute collide()
    pthread_barrier_wait(...);
  }
}
```

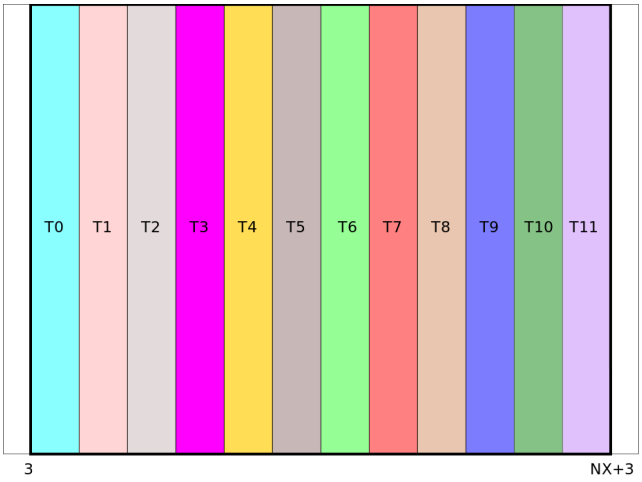


Figure 5: Each thread processes a slice of the sub-lattice, in this example 12 threads are used.

5 BlueGene/Q

In this project we have ported our LBM code to the IBM BG/Q architecture and investigated the performance.
BG/Q is a massively parallel supercomputing system with a very good power efficiency (FLOPS/Watt). Since November 2010 it is No. 1 on the Green500 list and since July 2011 it is No. 1 and 2 with a huge distance to No. 3 [7].
This new architecture surmounts the preceding BlueGene/P architecture due to higher clock frequency, more cores per chip, memory, etc..
On BG/Q each Compute Card there is a chip with 16+1+1 cores (16 computational cores, 1 helper core designed to handle OS service and 1 redundant spare) each capable of running SMT threads. Each core has a 16+16 kB L1 data and instruction cache connected to a shared 32 MB L2 cache. Each core comprises a Quad FPU SIMD that can process 4-wide DP vector instructions.
A water-cooled Node Card comprises 32 Compute Cards and optical modules to interconnect Node Cards. The 5-D torus network links have a bandwidth of 2 GB/s per link. A rack hosts 32 Node Cards, i.e. 1024 Compute Cards, 16384 cores which can execute up to 65536 threads.
Porting the code to BG/Q was really easy to do. No changes were required to get the code running on BG/Q producing correct results. Exploiting this large number of threads and the 4-wide SIMD is the real challenge now to improve the performance of this first code.

6 Optimization on BG/Q

A first easy optimization is to increase the number of threads per node.
With BG/Q we can use from 1 to 64 threads per compute card, with a really good scaling of the performance for this particular application.
The next optimization step is to use the 4 available FPUs to perform up to 2×4 floating-point operations per clock cycle and core.
To use all 4 FPUs we need a suitable data-structure. This structure has to use **vector** data types where

we can pack four double-precision numbers. These data types can be processed by **QPX instructions** which are thus able to process four lattice-sites instead of one.
We combine the data at distance $NY/4$ in the same variables as shown in Fig. 6.

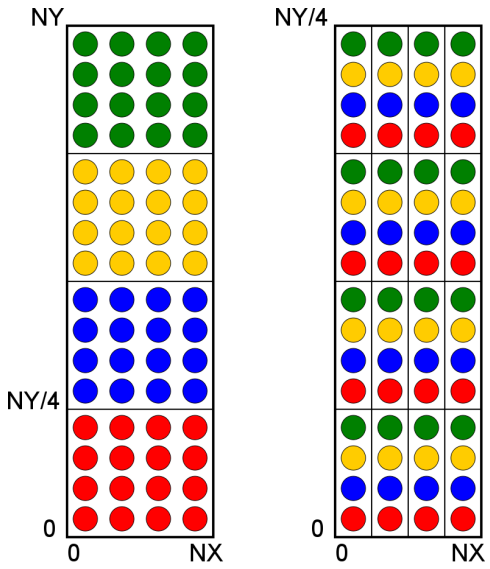


Figure 6: Cells at distance NY/4 are combined together in order to exploit the streaming vector instructions available in the processors.

Using this combine pattern we don't have to change all the code, because if we have some distance from the site combined in the same variable, the first neighbors are easy to find and compute, for example most of the blue sites need only other blue sites. In this case we don't need to change the code using QPX intrinsics, but we only change the data-type. Packing the variables inside the vector variables is straight-forward.
The new vectorized data-structure, called **v_pop_type**, looks as follows:

```
typedef struct {
  vectorType p[37];    // populations array
  vectorType u;        // horizontal velocity
  vectorType v;        // vertical velocity
  vectorType rho;      // density
  vectorType temp;     // temperature
  ...
} v_pop_type __attribute__ ((aligned (32)));

void init(v_pop_type *vf) {
  for ( i = 0 ; i < NX*NY/4 ; i++ )
    for ( k = 0 ; k < 37 ; k++ )
      for ( v = 0 ; k <  4 ; v++ ) {
        tf[v] = initialPopValue;
        vf[i].p[k] = vec_insert( tf[v], vf[i].p[k], v );
      }
}


void LBM_funct() {
```

```
    v_pop_type vf1[NX*NY/4], vf2[NX*NY/4];
    for ( step = 0 ; step < MAXSTEP ; step++ ) {
      stream (vf1, vf2);
      bc (vf1, vf2);
      collide (vf2, vf1);
    }
  }
```

To initialize this structure we have to insert the values in double precision in the right place inside the vectorized variable.
Operations on the vector variables are then translated by the XL compiler generating code which includes QPX instructions. In our code, we need to change only some operations in the streaming phase for the vectorization. During the stream() phase we need to swap the neighbors in Y+ and Y- when we are respectively in the upper and the lower rows of the lattice. For example, from Fig. 6 we can see that if we need the neighbor in $Y + 1$ of a red site we have to access the data of a blue site at the same position within the vector.
Using the vector instructions provides a significant performance speed-up, independently of the number of the threads and MPI processes used on the same node.

7 Results

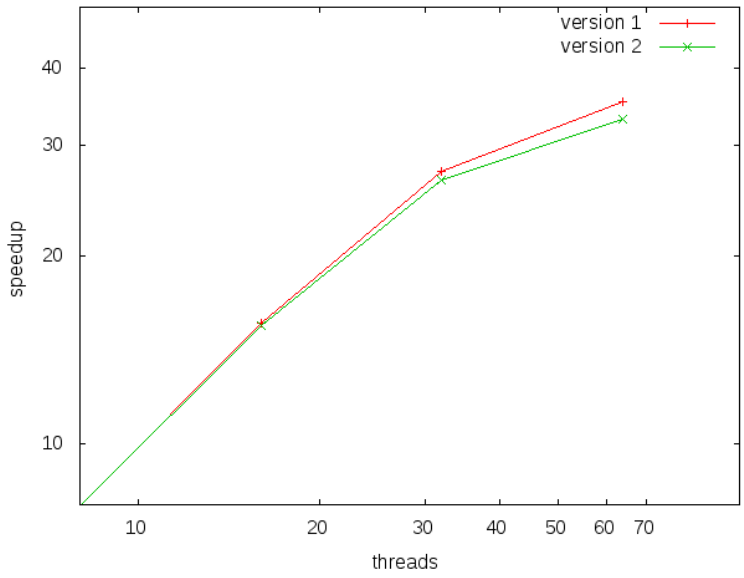


Figure 7: Plot of the performance as a function of the number of threads. Speedup is calculated as $\frac{t_1}{t_N}$ where t_N is the execution time with N threads and t_1 is the execution time with 1 thread

In this section we will present results obtained before and after the optimizations and results from multi-node runs. We start comparing the performance of the version without and with the vectorizations Fig. 7, where we call "version 1" the version without vectorization and "version 2" the vectorized one. We observe a significant speedup even if we use > 1 threads per core. Best performance is obtained

for 64 threads per node.
Now let us look at the performance of the code parallelized over several nodes, using 1 to 32 nodes with 4 MPI processes per node and 16 threads per process. We investigated both strong scaling (see Fig. 8) and weak scaling (see Fig. 9).

8 Conclusion

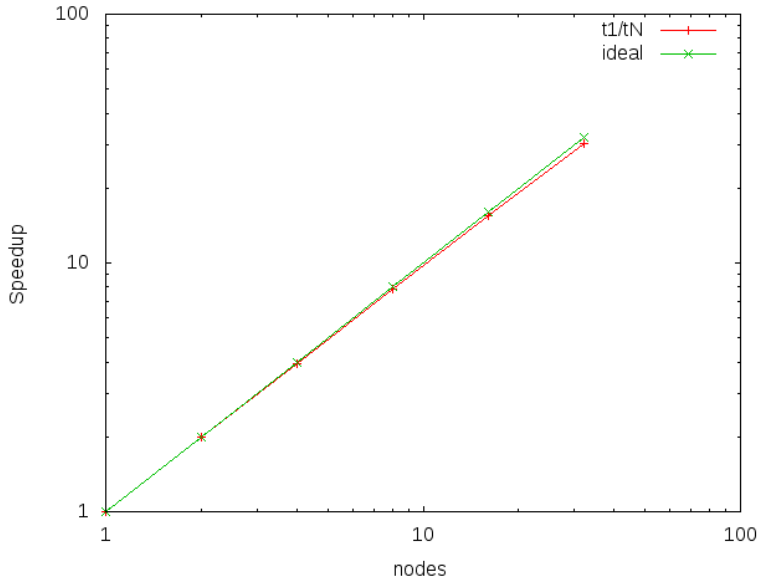


Figure 8: Strong scaling speedup, calculated as $\frac{t_1}{t_N}$. The global problem size is fixed, i.e. when we increase the number of compute nodes the problem size per node decreases.

In this report we describe the implementation, the porting and the optimization of a multi-phase LB code in 2D on the new massively parallel BlueGene/Q architecture. For this application, the inter-node parallelization is quite easy to implement, but a careful optimization is necessary to perform the intra-node parallelism. Starting from a code optimized for x86 architectures porting had been very easy. In fact, the code hardly required any particular changes.
We obtained the following results:

- We have not yet reached the performance we expected to reach on this architecture, because:
  - We used a prototype version of the system which does not run at target performance, e.g. only 1 out of 2 memory controllers was populated.
  - The algorithm used here has room for further improvements, e.g. we can merge the `stream()` and the `collide()` phases in a single phase. This would allow to reduce the execution time considerably.
  - Communication can most likely be improved by optimizing the logical mapping of the nodes onto the torus network.

- The vectorization of the code is relatively easy to perform. Using the **XL** compiler we need only to change the data structures. For this code we do not need to use QPX intrinsics since the compiler knows how to process vector data types.
- The intra-node scaling is surprisingly good. The current code strongly benefits from the 4-way SMT: Increasing the number of threads per core from 1 to 4 leads to a significant performance increase with an almost linear speed-up until 32 threads per node. Best performance is achieved using 64 threads per node.
- The scaling inter-node from 1 to 32 compute cards is really good, almost perfect.

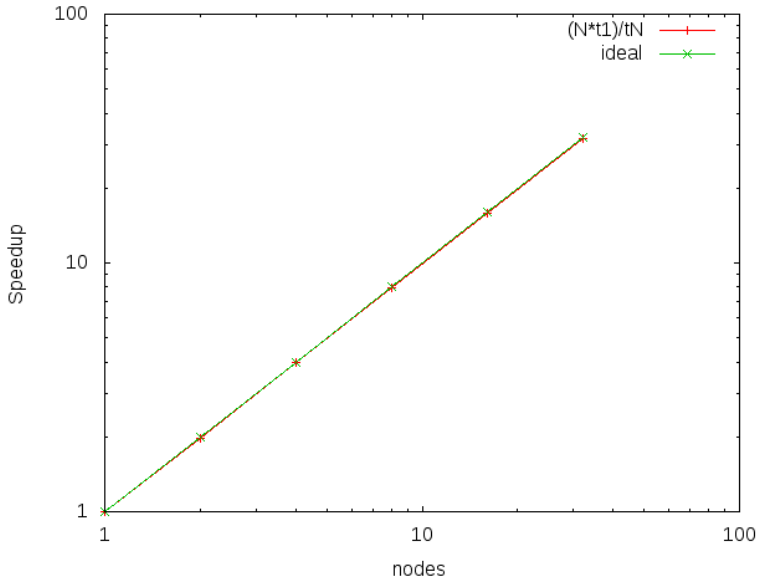


Figure 9: Weak scaling speedup, calculated as $\frac{Nt_1}{t_N}$. Here the problem size per node is fixed, i.e. when we increase the number of compute nodes the global size of the problem increases.

References

1. S. Succi, The Lattice Boltzmann Equation for Fluid Dynamics and Beyond, Oxford University Press (2001).
2. L. Biferale, et al., Lattice Boltzmann fluid-dynamics on the QPACE supercomputer, ICCS Proc. 2010, Procedia Computer Science, 1 (2010) 1075:1082.
3. L. Biferale et al., Lattice Boltzmann Method Simulations on Massively Parallel Multi-core Architectures, high performance computing symposium 2011, HPC 2011 (2011) 73-80.
4. L. Biferale et al., Optimization of Multi-Phase Compressible Lattice Boltzmann Codes on Massively Parallel Multi-Core Systems, Procedia Computer Science, ICCS (2011) 994-1003, Anno: 2011
5. M. Sbragaglia et al., Lattice Boltzmann method with self-consistent thermohydrodynamic equilibria, J. Fluid Mech., 628 (2009) 299
6. A. Scagliarini et al., Lattice Boltzmann methods for thermal flows: Continuum limit and applications to compressible Rayleigh-Taylor systems, Phys. Fluids, 22 (2010) 055101.
7. http://www.green500.org/lists/2011/06/top/list.php


JÜLICH
FORSCHUNGSZENTRUM